DB2 (Database 2)

History

IBM's first database is IMS. This is a hierarchical database. IBM introduces its second base on relational concepts in 1980s and it is called as Database 2 (DB2).

Advantage of DBMS over File Management System

1.Increased Independency

If a new field is added to a file, the layout of the file should be changed in all the programs with new field, though the programs do not use this field. But if a new column is added to a table, it does not need any program change, as long as the programs do not need this field. Thus programs using Databases are developed independent of physical structure.

2.Less Redundancy

In file system, the same data is stored in multiple files to meet the requirements of different applications. In DB2, we can store the common information in one table and make all the applications to share it. By means of centralized control and storing the fact at the right place and only once, data redundancy is reduced.

3.Increased Concurrency and Integrity

Concurrency allows more than one task to share a resource concurrently. Integrity is defined as validity or correctness of data. In file system, if a file is allocated with OLD disposition, then no other program can use it. DB2 allows more than one task to use the same table in update mode. At the same time, integrity of data is ensured by its sophisticated locking strategy.

4.Improved Security

In file system, all the records in the file are subjected to a specific access right. The access cannot be controlled at field level or set level. But in DB2, we can restrict access on column level and set level.

Types of Database

Hierarchical	Relation between entities is established using parent-child relationship-Inverted Tree Structure-This is the first logical model of DBMS. Data stored in the form of SEGMENTS. Example: IMS
Network	Any entity can be associated with any other entity. So distinguishing between parent and child is not possible. This is a linked structure model. Data stored in RECORD and different record types are linked by SETS. Example: IDMS
Relational	In mathematical terms, relation is TABLE. Data is stored in the form of tables consists of Tuples (rows) and attributes (columns). Example: DB2

DB2 Objects

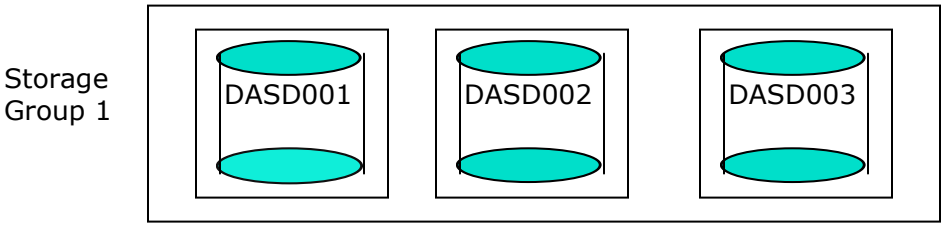
There are two types of elements or objects in DB2.

System Objects: objects that are controlled and used by DB2.
Example: DB2 Directory, DB2 catalog, Active & Archive logs, Boot Strap Dataset (BSDS), Buffer Pools, Catalog visibility Database and Locks.

Data Objects: Objects that are created and used by the users.
Example: Tables, Indexes, Views, Table spaces, Databases, Storage Groups.

DB2 Object-Storage Group

Storage group is a collection of direct access volumes, all of the same type. DB2 objects are allocated in Storage group. A storage group can have maximum 133 volumes.

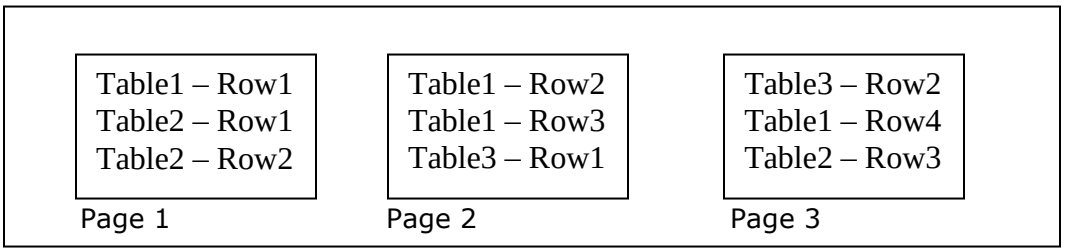


DB2 Object-Table Space

Table Spaces are the physical space where the tables are stored. There are three types of table spaces.

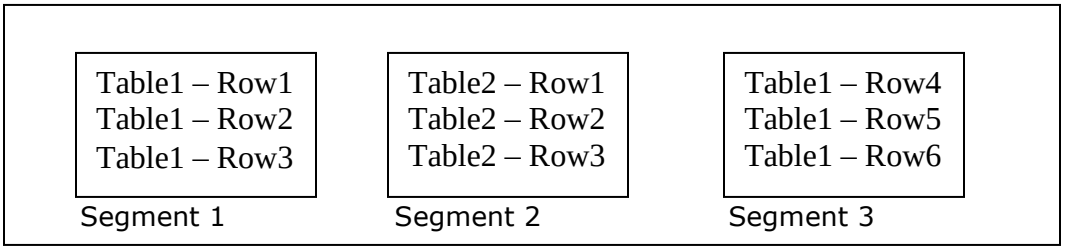
Simple Table Space:

More than one table is stored in the table space and single page can contain rows from all the tables.



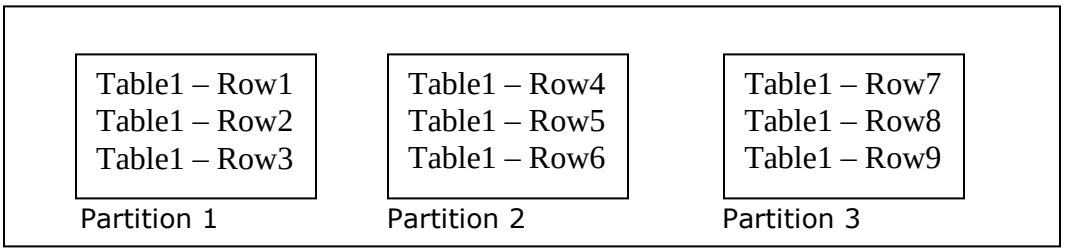
Segmented Table Space:

Segment is logically contiguous set of n pages where n is defined by SEGSIZE parameter of TABLESPACE definition. Tables are stored in one or more segments. Mass delete and sequential access are faster in Segmented type table space. Reorganization will restore the table in its clustered order. Lock table command locks only the table and not the table space.



Partitioned Table Space:

Only one table can be stored. 1-64 partitions are possible in table space. NUMPART of TABLESPACE definition decides the partitions. It is partitioned with value ranges for single or combination of columns and these columns cannot be updated. Individual partitions can be independently recovered and reorganized. Different partitions can be stored in different groups.



DB2 Object-Database

It is not a physical object. It is a logical grouping consists of Table spaces, Index spaces, Views, Tables etc. Whenever you create an object you have to explicitly say which Database it belongs to or DB2 implicitly assigns the object to the right database.

In the creation of table, we should say in which database and in which table Space we are housing the table (explicitly). In creating an index, we mention only the table name. DB2 decides the database from the base table name.

It can also be defined as a unit of start and stop. There can be a maximum of 65,279 databases in a DB2 Subsystem.

DB2 Object - Table

Table is a structure consists of number of columns and rows. It is created using the DDL CREATE TABLE.

Physical hierarchy:

ROWID is a pointer to data record in the page.
It is a 4byte field. (3 bytes for page and 1 byte for line in the page)
Page consists of 1 to 127 data records. It is the unit of I-O.
Table space consists of pages and it is stored in STOGROUP.
Database consists of one or more table space and other related objects.

DB2 Object- Index

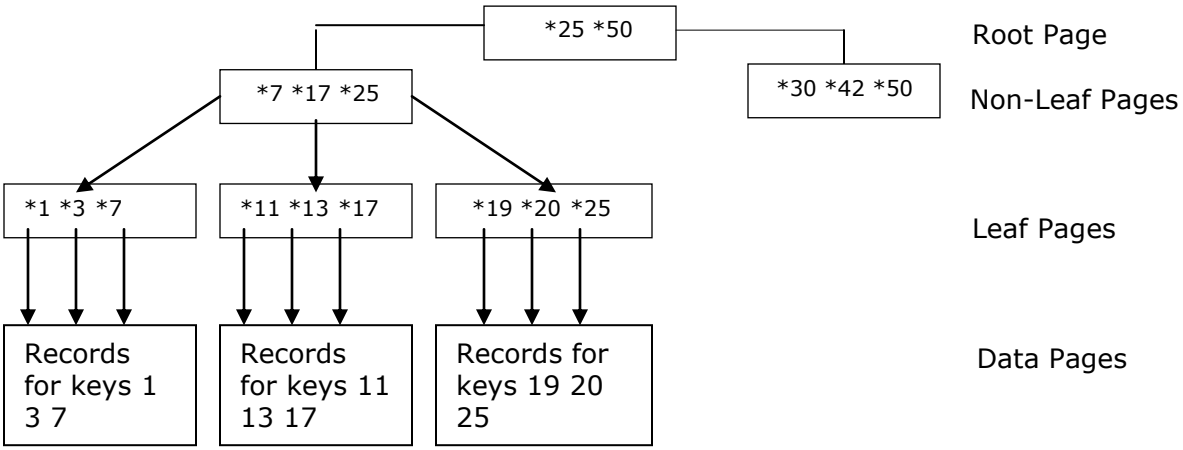
Generally, Index is used to access a record quickly. There are two types of Indexes: Clustering and Non-Clustering. If there is a Clustered index is available for a table, then the data rows are stored on the same page where the other records with similar index keys are already stored on. There can be only one cluster index for a table. Refer the diagram in the next page for better understanding of clustered and non-clustered indexes.

Understanding: Root Page, Leaf Page and Non-Leaf page

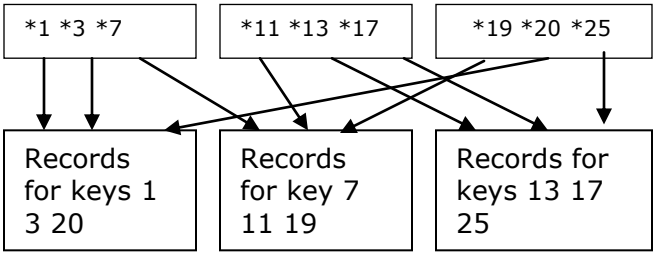
Referring to VSAM basics, sequence set has direct pointers to control interval and last level indexes have pointers to sequence set and higher-level indexes have pointers to lower level indexes and there is one root index from where the B-tree starts.

Similarly Leaf-pages contain full keys with row-ids that points to data page. (Row id has page number and line number). Non-leaf pages have pointers to leaf pages or lower level non-leaf pages and the root page has pointer to first level non-leaf pages.

View of clustered index – Diagram



View of Non-clustered index – Diagram (non-leaf pages and data pages are shown)



DB2 Object-Index Space

Unlike table spaces, Index spaces are automatically created when indexes are created. Indexes are stored in index spaces.

DB2 Object - Alias & Synonym

Alias and Synonym are alternate name for a table. The main differences between them are listed below:

SYNONYM	ALIAS
Private object. Only the user who created it, can access it.	Global object. Accessible by anyone.
Used in local environment to hide the high level qualifier.	Used in distributed environment to hide the location qualifier.
When the base table is dropped, associated synonyms are automatically dropped.	When base table is dropped, aliases are not dropped.
SYSADM authority is not needed.	To create an alias, we need SYSADM authority or CREATEALIAS privilege.

DB2 Object-View

Views provide an alternative way of looking at the data in one or more tables. A view is a named specification of a result table. For retrieval, all views can be used like base tables. Maximum 15 base tables can be used in a view.

Advantages of view

1.Data security: Views allows to set-up different security levels for the same base Table. Sensitive columns can be secured from the unauthorized Ids.

2.It can be used to present additional information like derived columns.

3.It can be used to hide complex queries. Developer can create a view that results from a complex join on multiple tables. But the user can simply query on this view as if it is a separate base table, without knowing the complexity behind the building.

4.It can be used for domain support. Domain identifies a valid range of values that a column can contain. This is achieved in VIEW using WITH CHECK OPTION.

Read Only Views

Views on which INSERT, UPDATE and DELETE operations cannot be carried out, are called non-updateable views or read only views.

Characteristics of Updateable views

1.It should be derived from one base table and should not contain derived columns.
2.Views should not contain GROUP BY, HAVING and DISTINCT clauses at the outermost level.
3.Views should not be defined over read-only views.
4.View should include all the NOT NULL columns of the base table.

DB2 Object- Catalog

It is a data dictionary for DB2, supporting and maintaining data about the DB2 environment. It consists of 54 tables in the system database DSNDB06.
The data in DB2, about the data in the tables are updated when RUNSTATS utility runs. The information in the DB2 catalog table can be accessed using SQL.

DB2 Object- Directory

This is second data dictionary of DB2 and used for storing detailed, technical information about the aspects of DB2's operation. It is for DB2 internal use only.
It cannot be accessed using SQL statements. It is stored in the system database DSNDB01.

DB2 Object- Active and Archive Logs

DB2 records all the data changes and significant events in active logs as and when they occur. In case of failure, it uses this data to recover the lost information. When active log is full, DB2 copies the content of active log to a DASD or magnetic tape data set called archive log.

DB2 Object- Boot Strap Dataset

It stores the inventory of all-active and all-archive logs. During installation of DB2, two BSDS are created and kept in different volumes. BSDS datasets are VSAM KSDS.

DB2 Object-Buffer Pools

Buffer Pools are virtual storage area, which DB2 uses for its temporary purpose. There are fifty 4K buffer pools and ten 32K buffer pools in DB2.
DB2 Default buffers are BP0, BP1, BP2 and BP32.

Program Preparation

SQL (Structured Query Language) is the language that is used to Retrieve / update / delete DB2 data. SQL statements are embedded into COBOL Program within the scope of 'EXEC SQL and END-EXEC.'

DB2 Program is first feed to DB2 Pre compiler that extracts the DB2 statements into DBRM and replace the source program DB2 statements with COBOL CALL statements. This modified source is passed to COBOL compiler and then link editor to generate load module. During pre compilation, time stamp token is placed on modified source and DBRM.

On the other side, DBRM undergoes binding process and DB2 Optimizes chooses the best access path for the extracted SQL statements and store it in PLAN.

Plan and Package

Binding process can happen in two stages, BIND PACKAGE and BIND PLAN. One DBRM is created for one program. If the main program calls n number of Sub-programs, then there will be n DBRMS in addition to main program DBRM.

These n + 1 DBRM can be directly feed to BIND PLAN to produce a single PLAN or create m number of intermediate packages each formed by one or more DBRM. This m numbers of packages are then feed to BIND PLAN step that produces PLAN. Package is not executable but Plan is executable. To run a DB2 Program, Plan is mandatory.

Advantages of Package:

1.When there is a change in sub program, it is enough to recompile the subprogram and create the PACKAGE for that sub program. There is no need for BIND PLAN.

2.Lck options and various bind options can be controlled at sub-program level.

3.It avoids the cost of large bind.

4.It reduces Rebound time.

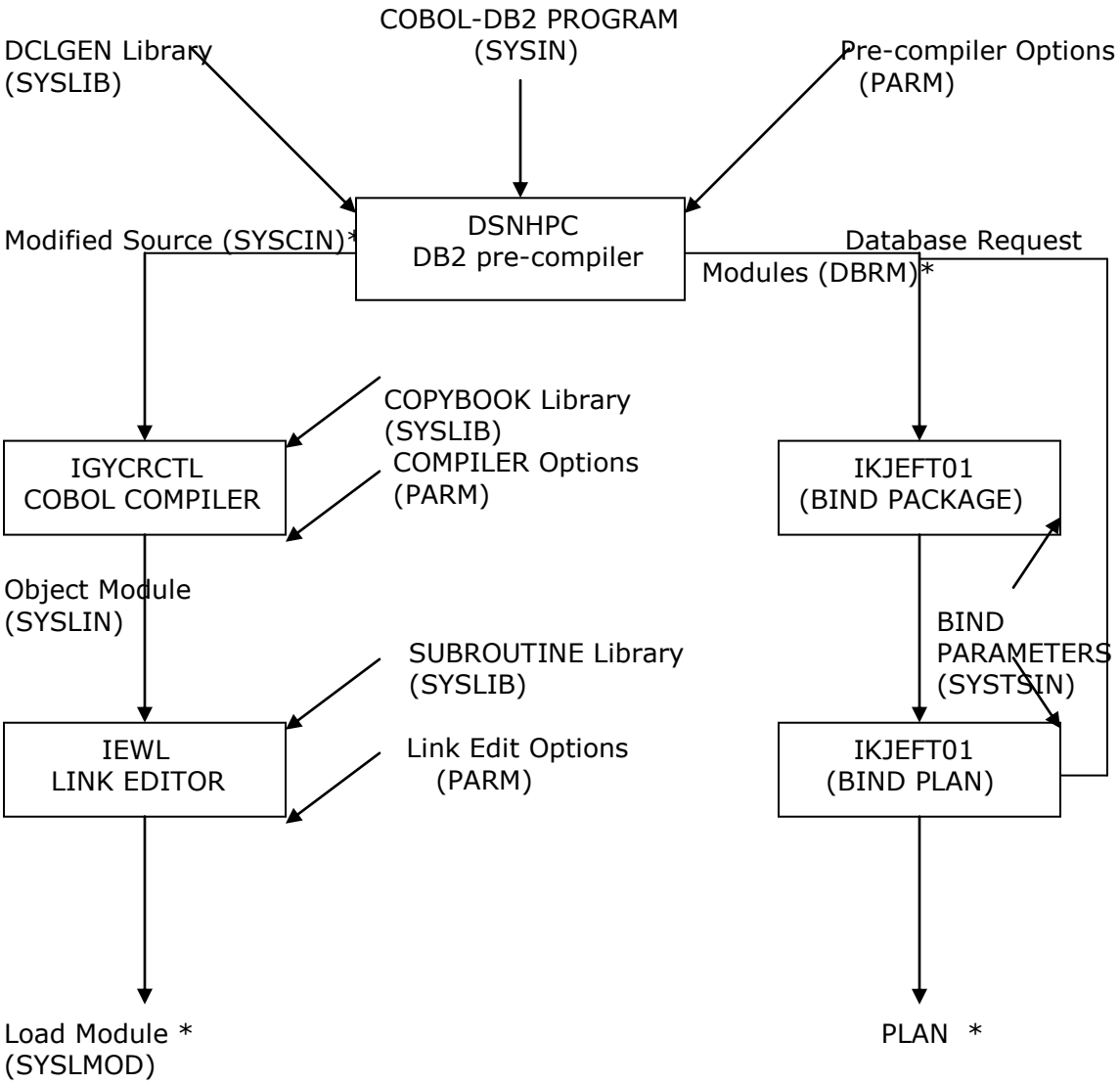
5.Versioning: Same package can be bounded with multiple programs with different collection IDs.

Execution of DB2 Program

DB2 Program is executed using terminal monitor program IKJEFT01. It can be executed using IKJEFT1A or IKJEFT1B also. They are alternate entry points of IKJEFT01.DB2 region name, program to run, DB2 plan and PARM (if needed) are provided in the SYSTSIN card of IKJEFT01.

```
//STEP EXEC PGM=IKJEFT01
//SYSPRNT DD SYSOUT=*
//SYSTSIN DD *
  DSN SYSTEM(DST2)                          == > In DB2 test region DST2
  RUN PROGRAM(pgmname) PLAN(planname) == > Execute the program
  LIB('loadlibrary)                         == > whose load module is here!
  END
/*
```

Program Preparation in Detail



* Time Stamp token is passed from pre-compilation stage to Load module and Plan.
When first time the plan is accessed, the time stamp of plan is verified against the time stamp of load module. If they match, then table will be accessed using the access path in PLAN and proceed further. If they don't match, then the program abnormally ends with SQLCODE of –818. It means the load module and plan are out of sync. It usually occurs if we modify the program and pre-compile, compile and link edit but did not bind the plan with latest DBRM produced in pre-compilation stage.

There is a modification in COBOL part. There is no change in any of the DB2 statements in the program. Is binding necessary?

Yes. It is necessary. For COBOL changes, we have to compile and link edit. For compilation we need pre-compiled source. So new time stamp is transferred to your load module from the pre-compilation stage. If we don't BIND it, timestamp mismatch is obvious and the program will abnormally end.

Can this time stamp check be avoided?

If the program pre-compiled with LEVEL option, then there won't be time stamp verification. But this option is not generally recommended.

Bind Card

BIND PACKAGE sub command is used to bind a DBRM to a package and BIND PLAN is used to bind a DBRM to a plan or group of packages to a PLAN.

Sample Bind Card:

```
//BIND EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
  DSN
  BIND PLAN(EMPPLAN)
  MEMBER(EMPDBRM)
  VALIDATE(BIND)
  ISOLATION(CS)
  RELEASE(COMMIT)
  OWNER(SMSXL86)
  LIB('SMSXL86.DB2.DBRMLIB')
/*
```

Bind Parameter – MEMBER and LIBRARY

DBRM to be bound is given in MEMBER and the partitioned dataset containing the DBRM is given in LIB. During BIND Package, if an existing package is used to create the new package then, COPY should be mentioned instead of MEMBER and LIBRARY.

COPY(collection-id.package-id)

COPYVER(version-id) – version of the copy to be used. Default is empty if omitted.

Bind Parameter – PKLIST

PKLIST is a BIND parameter of BIND PLAN. Packages to be connected with PLAN are named here. PKLIST (COLL1.*) bound all the packages with collection ID COLL1 to the PLAN.

Package Naming – Collection-ID and Versioning

Packages have three qualifiers. Location ID is used in distributed environment. Collection ID is used for grouping of packages.

Naming Syntax: Location-id. Collection-id. Package-name

VERSION. With packages, the pre-compiler option VERSION can be used to bind multiple versions of the programs into package.

PROG-------- > Pre-compile ----------→ BIND Package --------→COLL1.PROG.TEST
With VERSION(TEST)

-------- > Pre-compile ----------→ BIND Package --------→COLL1.PROG.PROD
With VERSION(PROD)

Bind Parameter – ACTION

Package or Plan can be an addition or replacement. Default is replacement. REPLACE will replace an existing package with the same name, location and collection. REPLVER(version-id) will replace a specific version of an existing package. ADD – Add the new package to SYSIBM.SYSPACKAGE.
Syntax: ACTION(ADD|REPLACE)

Bind Parameter – Isolation Level

The isolation level parameter of the BIND command describes to what extent a program bound to this package can be isolated from the effects of other programs running. This determines the duration of the page lock.

CURSOR STABILITY (CS)– As the cursor moves from the record in one page to the record in next page, the lock over the first page is released (provided the record is not updated). It avoids concurrent updates or deletion of row that is currently processing. It provides WRITE integrity.

REPEATABLE READ (RR) – All the locks acquired are retained until commit point. Prefer this option when your application has to retrieve the same rows several times and cannot tolerate different data each time. It provides READ and WRITE integrity.

UNCOMMITTED READ (UR) – It is also known as DIRTY READ. It can be applied only for retrieval SQL. There are no locks during READ and so it may read the data that is not committed. Highly dangerous and use it when concurrency is your only requirement. It finds its great use in statistical calculation of the large table and data-warehousing environment.

Bind Parameter – ACQUIRE and RELEASE

ACQUIRE(USE) and RELEASE(COMMIT) – DB2 imposes table or table space lock when it executes an SQL statement that references a table in the table space and it release the acquired lock on COMMIT or ROLLBACK. This is the default option and provides greater concurrency.

ACQUIRE(USE) and RELEASE(DEALLOCATE) – Locks table and table spaces on use and releases when the plan terminates.

ACQUIRE(ALLOCATE) and RELEASE(DEALLOCATE) – When DB2 allocates the program thread, it imposes lock over all the tables and table spaces used by the program. This option avoids deadlocks by locking your source at the beginning but it reduces concurrency.

ACQUIRE(ALLOCATE) and RELEASE(COMMIT) is NOT ALLOWED. This increases the frequency of deadlocks.

Bind Parameter – SQLERROR

It says whether to create the package in case of SQL errors.
SQLERROR (NOPACKAGE) – Default - If there is any SQL error, package will not be created.
SQLERROR (CONTINUE) – Create the Package even if SQL errors are encountered.

Bind Parameter – VALIDATE

It controls the handling of 'object not found' and 'not authorized' errors. Default is VALIDATE (RUN) – If all objects are found and all privileges are held, then don't check once again at execution time. If there is any 'privilege issue' or 'object not found' error, then produce WARNING messages, bind the package and check authorization and existence at execution time.

VALIDATE (BIND) – If there is any privilege or object-not-found issue, then produce ERROR message and create the package only if SQLERROR(CONTINUE) is coded in bind card. The statement in error will not be executable.

BIND Parameter – EXPLAIN

EXPLAIN(YES) loads the access path selected by the optimizer in PLAN_TABLE. EXPLAIN(NO) is default and it wont load any such details in PLAN_TABLE.

BIND Parameter – QUALIFIER

Qualifiers are added at the bind time to the objects used in the program. It is suggested NOT to use any qualifiers in the program and use this parameter during BIND so that the program can be easily promoted from one region to next region without any change in program.

REBINDING

When SQL statements are not changed but a new index is added or RUNSTATS is run, then it is advisable to do a REBIND for the best possible access path. REBIND PLAN is cheaper than BIND with ACTION (REPLACE).

SYSTSIN CARD should be

```
REBIND PLAN(PLAN-NAME)
VALIDATE(BIND)
…..
```

Host Variables

DB2 is separate subsystem. As DB2 data are stored in external address space, you cannot directly modify or read from your program. During read, the DB2 values are retrieved into your working storage variables. During update, the DB2 columns are updated using your working storage variables. These working storage variables used for retrieving or updating DB2 data should be of same size of DB2 columns. They are called as host variables as they are defined in the host language (COBOL). Host variables are prefixed with colon (:) in the embedded SQL.

DCLGEN (Declaration Generator)

DCLGEN is a DB2 utility. If we provide table details, it will generate DB2 Structure and host variable structure of the table. It can also generate NULL indicators for the NULL columns. This output is stored in a PDS member. These members are included in the program using the INCLUDE <pds-member>. INCLUDE is a pre-compiler statement and so it should be coded within the scope of EXEC SQL and END-EXEC.

DCL generated host variable names are same as DB2 column names. As underscore is not a valid character for variable names in COBOL, it is replaced by hyphen in host variable generation of DCLGEN. Prefix and suffix can be added to all the variables while creating the DCLGEN copybook.

Host variables can be declared in working storage section. But they cannot be stored in copybooks as other file layouts.

INCLUDE is expanded during pre-compilation time but COPY is expanded during compilation and so declaration of host variable in copybook will produce errors during pre-compilation.

DECLARE TABLE, that is table structure of DCLGEN is NOT really needed for pre-compilation. But if it is used, then any misspelled table name, column names are trapped in pre-compilation stage itself.

DB2 need not be up during pre-compilation. As pre-compiler just extracts the DB2 statements and produce DBRM, it does not need DB2 connection. But DB2 region should be up for binding as the optimizer builds access path in this stage from catalog information.

DCLGEN Panel (Option 2 of DB2I)

```
 DSNEDP01                  DCLGEN                              SSID: DSN
 ===>

 Enter table name for which declarations are required:
  1  SOURCE TABLE NAME ===> DSN8410.EMPTABLE    (Unqualified table name)
  2  TABLE OWNER       ===>                                 (Optional)
  3  AT LOCATION ..... ===>                                 (Optional)

 Enter destination data set:          (Can be sequential or partitioned)
  4  DATA SET NAME ... ===> 'SMSXL86.DB2.INCLUDE(EMPTABLE)'
  5  DATA SET PASSWORD ===>           (If password protected)

 Enter options as desired:
  6  ACTION .......... ===> ADD       (ADD new or REPLACE old declaration)
  7  COLUMN LABEL .... ===> NO        (Enter YES for column label)
  8  STRUCTURE NAME .. ===>                                   (Optional)
  9  FIELD NAME PREFIX ===>                                   (Optional)
 10  DELIMIT DBCS .... ===> YES       (Enter YES to delimit DBCS identifiers)
 11  COLUMN SUFFIX ... ===> NO        (Enter YES to append column name)
 12  INDICATOR VARS .. ===> NO        (Enter YES for indicator variables)


 PRESS: ENTER to process    END to exit      HELP for more information
```

DSNE905I EXECUTION COMPLETE, MEMBER EMPTABLE ADDED

DB2 Data Types and equivalent host variables.

DB2 Column	Bytes	COBOL PIC Clause	Bytes
SMALLINT	2	PIC S9(04) COMP	2
INTEGER	4	PIC S9(09) COMP	4
DECIMAL (p, q) (P should be less than 32)	Int (P/2)	PIC S9(p-q)V9(q)	Integer ((P+Q)/2 +1)
DATE	4	PIC X(10)	8
TIME	3	PIC X(08)	6
TIMESTAMP	10	PIC X(26) yyyy-mm-dd-hh.mm.ss.nnnnnn	26
CHAR(n) (n=1 to 254)	N	PIC X(n)	N
VARCHAR(n) (n=0-4046)	N	01 WS-COLUMN. 49 WS-COLUMN-LENGTH PIC S9(04) COMP 49 WS-COLUMN –TEXT PIC X(n)	N+2

Limit of Date: Jan 1st 1 ad – Dec 31st 9999 AD
Limit of Time: 000000 – 240000
Limit of Time Stamp: 00010101000000000000 – 00001231240000000000

SQLCA

SQLCA is SQL Communication area. It is a DB2 copybook that should be included in all the DB2 programs. The fields declared in the copybook are updated for every DB2 query. The Length of SQLCA is 136. The most important field in this copybook is SQLCODE. The success or failure of DB2 query can be checked in this field.

Value of 0 in SQLCODE indicates the successful operation.
Positive value in SQLCODE indicates the completed operation with some exception
Negative value in SQLCODE indicates the unsuccessful operation.

So it is common programming practice that the SQLCODE of every query must be verified for valid values after the execution of the query. If the SQLCODE value is not acceptable, then the control is transferred to ABEND routine that would end the program with proper error messages.

SQLERRD (3) contains the number of rows affected by a DB2 INSERT/DELETE operation. If the operation is DELETE, this contains number of rows deleted after the execution of the query.

WHENEVER statement

WHENEVER is a error-trapping statement that directs the processing to continue or branch to an error handling routine based on the SQLCODE returned for the statement. When it processed, it applies to all the subsequent SQL statements issued by the application program.

```
EXEC SQL
     WHENEVER condition action
END-EXEC.
```

Conditions can be NOT FOUND, SQLWARNING or SQLERROR.
Action can be either CONTINUE or GO TO.

NOT FOUND is TRUE if SQLCODE is 100.
SQLWARNING is TRUE if SQLCODE is greater than zero but not equal to 100
SQLERROR is TRUE if SQLCODE is less than zero.

It is almost always safe to code specific SQLCODE checks after each SQL statement and process accordingly. Avoid using WHENEVER.

DSNTIAR – SQLCA Formatter

DSNTIAR is an IBM supplied program, which would display the error messages in a formatted way. Most of the installations call this program in their error routine.

```
CALL 'DSNTIAR' USING SQLCA, ERROR-MESSAGE, ERROR-TEXT-LEN.

01 ERROR-MESSAGE.
   05 ERROR-LEN PIC S9(04) COMP VALUE +1320.
   05 ERROR-TEXT PIC X(132) OCCURS 10 TIMES.

77 ERROR-TEXT-LEN PIC S9(09) COMP VALUE +72.
```

Components of DB2

DB2 internal structure is very complex and contains large number of components. But from a high-level point of view, DB2 can be considered to have four major components.

System Services component: Supports system operation, operator communication, logging and similar function.

Locking services component: Provides the necessary controls for managing concurrent access to data.

Database services component: Supports the definition, retrieval and update of user and system data.

Distributed data facility component: Provides DB2's distributed database support.

Before get into SQL, let us briefly see the sub- components of Database services component.

Database services component

It has five sub-components in it. We have already discussed the functions of two components – Pre-compiler and Optimizer (Bind)

Runtime supervisor:

It is resident in the main memory when the application program is executing. Its job is to oversee that execution. When the program requests some database operation to be performed, control first goes to the runtime supervisor, which uses the control information in the application plan to request the appropriate on the part of the Data Manager.

Data Manager (DM):

It performs all the normal access method functions – search, retrieval, update etc. It invokes other system components in order to perform detail functions such as locking, logging etc.

Buffer Manager (BM):

It is responsible for transferring the data between external storage and virtual memory. It uses sophisticated techniques such as 'read-ahead-buffering' and 'lock-aside-buffering' to get the best performance out of the buffer pools under its control to minimize the amount of physical i/o actually performed.

Catalog and Directory are also part of Database services component only.

SQL

SQL is fourth generation language. The definition of fourth language is 'Tell the need. System will get it done for you `. There is no need to instruct the method to get it.

For example, I want list of employees working currently in Malaysia. If the data is in file, then I have to declare, define, and Open the file. Then I should read all the records into my working storage one by one and if their location is Malaysia, I display them. Finally I close the file.

If the data is in DB2, I would simply write a query (SELECT * FROM table where location='Malaysia'') and DB2 optimizer do everything for me in identifying the best access path. No worries of DECLARE, DEFINE, OPEN, READ and OMIT stuffs. That's the power of fourth generation language and that makes the programmer life easy!!

SQL has three kinds of statements.

DDL–Data Definition Language Statements	CREATE ALTER DROP
DML–Data Manipulation Language Statements	SELECT INSERT UPDATE DELETE
DCL–Data Control Language Statements	GRANT REVOKE

DDL-CREATE

This statement is used to create DB2 objects.

General Syntax CREATE object object-name parameters

Table creation Sample
```
CREATE TABLE table-name
 (Column definitions,
   Primary Key definition,
   Foreign Key definition,
   Alternate Key definition,
   LIKE table-name /View-name,
   IN DATABASE-NAME.TABLESPACE-NAME)
```

DDL-Column Definition

Define all the columns using the syntax:

Column-name Data-type Length (NULL/NOT NULL/ NOT NULL WITH DEFAULT)

Data-types are already explained in DCLGEN section.

- Column-name can be of maximum length 30. It should start with an alphabet and it can contain numbers and underscore.
- NULL means UNKNOWN, if the value is not supplied during an insertion of a row, then NULL will be inserted into this column.
- NOT NULL means, value for this column should be mentioned when inserting a row in the table. NOT NULL with DEFAULT means, if the value for this column is not supplied during an insertion, then based on type of the column default values will be moved into the table. Default values are listed in the below table.

Data- Type	Default Value
CHAR	Spaces
VARCHAR	Empty String (String of length 0)
DATE, TIME, TIMESTAMP	Current (DATE/TIME/TIMESTAMP)
INTEGER SMALLINT DECIMAL	Zero

DDL-Primary Key definition

Candidate key is defined as column(s) that uniquely identifies each row in the table. One of the candidate keys is defined as primary during the table creation. The main entity, about which the information is stored, is selected as primary key.

If the PRIMARY KEY is single column, then it can be coded along with column definition. If the primary key is composite key (more than one column), then it can be defined after the column definition. Primary key definition can be added or dropped later using ALTER table.

The definition of primary key is complete only after unique index is created on it. Until that time, the table is unavailable for use. If the primary key is added using ALTER command, then unique index on the key column(s) must already be available.

Syntax: PRIMARY KEY (column1, column2)

DDL-Foreign Key definition

Defining a foreign key establishes a referential constraint between the foreign key (Child Table) and the primary key (Parent table). The parent table of a referential constraint must have a primary key and a primary index.

There must be a value in parent table (primary key) for every non-null value in child table (foreign key). Foreign key can be defined along with column definition or separately after the definition of all the columns.

Syntax 1: (Along with column definition)
COLUMN1 CHAR(4) REFERENCES TABLE2 ON DELETE CASCADE/DELETE/SET NULL
Syntax 2: (After column definition)
FOREIGN KEY FKEY(COLUMN1,COLUMN2) REFERENCES TABLE2 ON DELETE/CASCADE/SET NULL

Delete Rule

Delete rule defines what action needs to be taken on child table when there is a deletion in primary table. The three possibilities are below:

CASCADE: Child table rows referencing parent table will also be deleted.
RESTRICT: Cannot delete any parent row when there is reference in child table.
SET NULL: SET the foreign key value as NULL when the respective primary row is deleted.

Insert and Update rules cannot be defined. But they can be implemented using TRIGGERS.

DDL-Alternate Key Definition

Candidate keys that are not defined as primary key are called alternate keys. This can be defined along with column definition or separately after the definition of all the columns using the keyword UNIQUE. Alternate key definitions CANNOT be added later using ALTER command.

Syntax 1: COLUMN1 CHAR(10) UNIQUE NOT NULL;
Syntax 2: UNIQUE(Column1)

DDL- Adding Constraints

A check constraint allows you to specify the valid value range of a column in the table. For example, you can use a check constraint to make sure that no salary in the table is below Rs10000, instead of writing a routine in your application to constrain the data.

CHECK clause and the optional clause CONSTRAINT (for named check constraints) are used to define a check constraint on one or more columns of a table. A check constraint can have a single predicate, or multiple predicates joined by AND or OR. The first operand of each predicate must be a column name, the second operand can be a column name or a constant, and the two operands must have compatible data types.

Syntax 1: ADD CHECK (WORKDEPT BETWEEN 1 and 100);
Syntax 2: ADD CONSTRAINT BONUSCHK CHECK (BONUS <= SALARY);

Although CHECK IS NOT NULL is functionally equivalent to NOT NULL, it wastes space and is not useful as the only content of a check constraint. However, later if you want to remove the restriction that the data be Non-null, you must define the restriction using the CHECK IS NOT NULL Clause.

DDL-Index creation

Table spaces should be created before creating the table. But index spaces are automatically created with creation of index. So create index command provides optional parameters of USING STOGROUP, PRIQTY, SECQTY, FREEPAGE and PCTFREE etc. for the allocation of index space. But usually we don't mention these parameters and allow the system to allocate it in the right place with respect to table space.

CREATE [UNIQUE] INDEX index-name on Table-name (column-name [asc/desc],..)
CREATE UNIQUE INDEX EMPINDEX ON EMPTABLE (EMPNO ASC)

Creating unique index on a NULL column is possible if there is only one NULL is available in that column.

DDL-Sample Storage group Creation

```
CREATE STOGROUP DSN8G41D
     VOLUMES (DBPK01, DBPK02)
      VCAT DB2CATZ;
```

Using the ALTER command, volumes can be added or removed at later stage using the command ADD VOLUMES and REMOVE VOLUMES.

DDL-Sample Database Creation

```
CREATE DATABASE DSN8D41A
      STOGROUP DSN8G410
      BUFFERPOOL BP0;
```

DDL-Sample Table space Creation

```
CREATE TABLESPACE DSN8S41D
      IN DSN8D41A
      USING STOGROUP DSN8G410
           PRIQTY 20
           SECQTY 20
           ERASE NO
      LOCKSIZE PAGE LOCKMAX SYSTEM
      BUFFERPOOL BP0
      CLOSE NO
      FREEPAGE 0 PCTFREE 5
      DSETPASS DSN8;
```

PRIQTY –Primary space to be allocated during table space creation. (KB)
SECQTY -Secondary space taken as amount of data in table space grows. (KB)
ERASE –Whether DB2 defined datasets are to be erased when table space is dropped.
LOCKSIZE – Indicates size of lock – Can be TABLESPACE, TABLE, PAGE, ROW or ANY. ANY means DB2 will decide the lock.
LOCKMAX – Indicates maximum number of page / row locks above which lock escalation occurs.
BUFFERPOOL – Buffer pool associated with the table space.
CLOSE – Indicates whether datasets associated with table space should be closed when there are no current users of table space.

FREEPAGE – Number of pages after which a empty page is available.
PCTFREE - Percentage of page to be left free for future inserts.

DDL-Sample Table Creation

```
CREATE TABLE DSN8410.EMP
        (EMPNO       CHAR(6)        NOT NULL,
         NAME        VARCHAR(50)  NOT NULL,
        WORKDEPT   CHAR(3),
        PHONENO     CHAR(4)        CONSTRAINT NUMBER CHECK
                    (PHONENO >= '0000' AND
                     PHONENO <= '9999'),
         BIRTHDATE DATE,
         SALARY      DECIMAL(9,2),
         PRIMARY KEY (EMPNO),
         FOREIGN KEY RED (WORKDEPT) REFERENCES DSN8410.DEPT
        ON DELETE SET NULL)
      IN DSN8D41A.DSN8S41D;
```

DDL-Sample View Creation

The general syntax of view definition is
CREATE VIEW view-name (column1,..) AS sub-query [WITH CHECK OPTION]

WITH CHECK OPTION ensures that all the inserts and updates made to the base table using this view satisfies the limit imposed by the where condition of the sub-query. (Domain Integrity)

```
CREATE VIEW DB2GROUP (EMPID EMPNAME SKILLSET)
AS SELECT EMPID, EMPNAME, SKILLSET
   FROM DSN8410.EMP
    WHERE SKILLSET = 'DB2' WITH CHECK OPTION;
```

This view shows name and id of all the employees who knows DB2. 'WITH CHECK OPTION' ensures that all the inserts made to base table using the view DB2GROUP, must have DB2 as SKILLSET. Any update-attempt to change the value of SKILLSET from DB2 to anything else using the view DB2GROUP is restricted.

DDL-ALTER TABLE

When you create a table using model table, the columns are defined in the new table as in the model table. But the PRIMARY, FOREIGN KEY definitions are not inherited. We have to define them using ALTER TABLE. Before defining primary key using ALTER TABLE, there should be unique index.

```
CREATE TABLE YDEPT
   LIKE DSN8410.DEPT;

CREATE UNIQUE INDEX YDEPTX
   ON YDEPT (DEPTNO);

ALTER TABLE YDEPT
  PRIMARY KEY(DEPTNO);
```

Using ALTER statement, alternate keys cannot be defined. Other than that we can add columns, constraints, checks & DROP primary key.

When a new column is added with NOT NULL WITH DEFAULT, then the value for the column for all the rows already exist in the table is filled based on the type of column:

1. Numeric –Zero
2. Char, Varchar -Spaces
3. Date -01/01/0001
4. Time -00:00 AM

Using ALTER command, we can extend the length of VARCHAR/CHAR items, switch the data-type of a column within character data-types (CHAR/VARCHAR);numeric data types(SMALLINT,INTEGET,REAL,FLOAT,DOUBLE,DECIMAL) and within graphical datatypes(GRAPHIC,VARGRAPHIC).

```
ALTER TABLE DSN8410.EMP
      ALTER COLUMN WORKDEPT
      SET DATATYPE CHAR(6);
```

DDL-DROP

DROP deletes the table definition in addition to all the rows in it. You also lose all synonyms, views, indexes, referential and check constraints associated with that table. Plans and Packages are deleted using the command FREE.

DROP Object Object-name.
DROP TABLE table-name
FREE PACKAGE(COLL1.*) - Deletes all the packages belongs to collection COLL1.

Data Manipulation Language statements

DML-SELECT

SELECT statement is the heart of all queries and it is specifically used for retrieving information. As a primary statement to retrieve the data, it is the most complex and most complicated DML statement used. Six different clauses can be used with a SELECT statement and each of these clauses has its own set of predicates. They are FROM-WHERE-GROUPBY-HAVING-UNION and ORDER BY.
Syntax: SELECT columns FROM tables
WHERE conditions
GROUP BY columns
HAVING conditions
ORDERBY columns
Up to 750 columns can be selected. 15 sub-queries can be coded in addition to one main query.

DML-FROM clause of SELECT

FROM clause is used, in conjunction with one or more references, to specify DB2 data manager where to retrieve data.

WHERE Clause

It is always followed by a search condition that contains one or more predicates that define how the DB2 DM is to choose the information that will be contained in the result dataset produced.
Any relational operator can be coded. (<>, >. < , = ,<=, >=)

DML-SELECT-WHERE- NOT Condition

NOT keyword can be used to select all the rows except the row identified with the search condition. Syntax: *WHERE NOT <condition>*.
'NOT' cannot be used directly with comparison operators. NOT = is invalid.

DML-SELECT-WHERE-LIKE condition

LIKE is used for pattern search. '%' is a wild card for any number of zero or more characters and '_' is used to indicate single unknown character.

If you want to look for % in the string, then you should define ESCAPE character. Ex: WHERE col1 LIKE 'MUTHU_+%SARA%' ESCAPE '+'

This will look for a string that starts with 'MUTHU_%SARA' where _ can be any character.

DML-SELECT-WHERE-BETWEEN and IN

Use BETWEEN to check the value of column is within the range. The lower and upper boundaries are INCLUSIVE. Use IN to check the value of column against a Values list.

DML-SELECT-WHERE-ANY/SOME/ALL

SELECT…WHERE COLUMN1 > ANY|ALL|SOME (Sub-Query)

ALL - If the value of COLUMN1 is greater than all the values return by sub-query, then only the outer table row will be selected. If no rows are returned by sub-query then all the rows of outer table will be selected.

ANY and SOME – If COLUMN1 value is greater than one of the values return by sub-query, then the outer table row will be selected.

DML-SELECT-WHERE-EXIST

This is useful when you want to just check whether one or more rows exist.
Syntax: WHERE EXISTS (sub-query)

Your main query will return result only if at least one row exists for your sub query. NOT EXISTS check for unavailability of any rows for the search condition in the sub query.

DML-SELECT-GROUP BY

Use GROUP BY for grouping rows by values of one or more columns. You can then apply column function for each group. Except for the columns named in the GROUP BY clause, the SELECT statement must specify any other selected columns as an operand of one of the column functions.

If a column you specify in the GROUP BY clause contains null values, DB2 considers those null values to be equal. Thus, all nulls form a single group.
You can also group the rows by the values of more than one column

DML-SELECT-HAVING (Subjecting Group to conditions)

Use HAVING to specify a search condition that each retrieved group must satisfy. The HAVING clause acts like a WHERE clause for groups, and contain the same kind of search conditions you specify in a WHERE clause.

The search condition in the HAVING clause tests properties of each group rather than properties of individual rows in the group.

DML-SELECT-UNION and UNION ALL

Using the UNION keyword, you can combine two or more SELECT statements to form a single result table. When DB2 encounters the UNION keyword, it processes each SELECT statement to form an interim result table, and then combines the interim result table of each statement. If you use UNION to combine two columns with the same name, the result table inherits that name.

You can use UNION to eliminate duplicates when merging lists of values obtained from several tables whereas UNION ALL retains duplicates.

DML-SELECT-ORDER BY

The ORDER BY clause is used to sort and order the rows of data in a result dataset by the values contained in the column(s) specified. Default is ascending order (ASC). If the keyword DESC follows the column name, then descending order is used. Integer can also be used in place of column name in the ORDER BY clause.

SELECT COL1,COL2,COL3 FROM TAB1 ORDER BY COL1 ASC COL3 DESC.
SELECT COL1,COL2,COL3 FROM TAB1 ORDER BY 1 ASC 3 DESC.

DML-SELECT-CONCAT

CONCAT is used for concatenating two columns with the string you want in between. To concatenate the last name and first name with comma in between,

```
SELECT LASTNAME CONCAT ',' CONCAT FIRSTNME
  FROM DSN8410.EMP;
```

COLUMN FUNCTION AND SCALAR FUNCTION

Column function receives a set of values from group of rows and produces a single value. SCALAR function receives one value and produces one value.

COLUMN FUNCTION:

Function	Returns--	Example
MAX	Maximum value in a column	MAX(SALARY)
MIN	Minimum value in a column	MIN(SALARY)
AVG	Average value of a column	AVG(SALARY)
SUM	Sum value of a column	SUM(SALARY)
COUNT	Number of selected rows	COUNT(*)

1. DISTINCT can be used with the SUM, AVG, and COUNT functions. The selected function operates on only the unique values in a column.
2. SUM and AVG cannot be used for non-numeric data types whereas MIN, MAX and COUNT can.

SCALAR FUNCTION:

Function	Returns---	Example
CHAR	String representation of its first argument	CHAR(HIREDATE)
DATE	Date derived from its argument	DATE('1977-12-16')
DAY	Day part of its argument	DAY(DATE1-DATE2)
DAYS	Integer representation of its argument	DAYS('1977-12-16') - DAYS('1979-11-02') + 1
MONTH	Month part of its argument	MONTH(BIRTHDATE) = 16
YEAR	Year part of its argument	YEAR(BIRTHDATE)=1977
DECIMAL	Decimal representation of its first argument	DECIMAL(AVG(SALARY),8,2)
FLOAT	Floating-point representation of its first argument	FLOAT(SALARY)/COMM
HEX	Hexadecimal representation of its first argument	HEX(BCHARCOL)
INTEGER	Integer representation of its argument	INTEGER(AVG(SALARY)+0.5)
HOUR	Hour part of its argument	HOUR(TIMECOL) > 12
MINUTE	Minute part of its argument	MINUTE(TIMECOL) > 59
SECOND	Second part of its argument	SECOND(TIMECOL) > 59
MICROSECOND	Microsecond part of its argument	MICROSECOND(TIMECOL) > 12

TIME	Time derived from its argument	TIME(TSTMPCOL) < '12:00:00'
TIMSTAMP	Timestamp derived from its argument or arguments	TIMESTAMP(DATECOL,TIMECOL)
LENGTH	Length of its argument	LENGTH(ADDRESS)
SUBSTR	Sub-string of a string	SIBSTR(FSTNAME,1,3)
VALUE	The first argument that is not null	VALUE(SMALLINT1,100) + SMALLINT2 > 1000

CURSOR

CURSOR is useful when more than one row of a table to be processed. It can be loosely compared with sequential read of file. Usage of cursor involves four steps.

1.DECLARE statement.

This statement DECLARES the cursor. This is not an executable statement.

Syntax: DECLARE cursor-name CURSOR [WITH HOLD] FOR your-query
[FOR UPDATE OF column1 column2 | FOR FETCH ONLY]

2.OPEN statement.

This statement just readies the cursor for retrieval. If the query has ORDER BY or GROUP BY clause, then result table is built. However it does not assign values to the host variables.

Syntax: OPEN cursor-name

3.FETCH statement.

It returns data from the results table one row at a time and assigns the value to specific host variables.

Syntax: FETCH cursor-name INTO
:WS-Column1
:WS-Column2

The number of INTO variables should be equal to SELECT columns of your DECLARE cursor statement. If FOR UPDATE OF clause is coded, then while fetching the row, the exclusive lock is obtained over the page of the row. FETCH Cursor is usually placed in the perform loop that will be executed until the SQLCODE is 100.SQLCODE will be set to 100 when end of cursor is reached.

If FOR UPDATE OF clause is given, then all the columns that are going to be updated using the cursor should be mentioned in this clause. If you don't mention FOR UPDATE OF clause, you can update any number of columns using WHERE CURRENT OF cursor-name but you will be getting the exclusive lock only during the UPDATE statement and there are chances for the row to be locked exclusively by another task in between your READ and UPDATE.

So it is suggested to use FOR UPDATE OF clause when you are using the cursor for update purpose.

4.CLOSE statement.

It releases all resources used by the cursor. If you don't close the cursor, COMMIT statement will close the cursor. To retain the cursor in OPEN stage during COMMIT, Use WITH HOLD option in the DECLARE statement of the CURSOR. This option will be effective only in batch environment. ROLLBACK close all the cursors including the cursors defined with 'WITH HOLD' option.

Syntax: CLOSE cursor-name.

Read Only Cursor

The cursor that cannot be used for any update purpose is called as READ ONLY CURSOR. For example, cursor defined with UNION, JOIN or GROUP BY cannot be used for update purpose.

DML-INSERT

INSERT statement is used to insert rows to the table. NOT NULL Column values should be supplied during INSERT. Otherwise INSERT would fail.

1.Specify values for columns of single row to insert.

```
INSERT INTO YDEPT (DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION)
  VALUES ('E31', 'DOCUMENTATION', '000010', 'E01', ' ');
INSERT INTO YEMP
  VALUES ('000400', 'RUTHERFORD', 'B', 'HAYES', 'E31',
    '5678', '1983-01-01', 'MANAGER', 16, 'M', '1943-07-10', 24000,
    500, 1900);
```

You can name all the columns or omit. When you list the column names, you must specify their corresponding values in the same order as in the list of column names.

2.Mass Insert. Another table or view contains the data for the new row(s).

```
INSERT INTO TELE
  SELECT LASTNAME, FIRSTNME, PHONENO
    FROM DSN8410.EMP
    WHERE WORKDEPT = 'D21';
```

If the INSERT statement is successful, SQLERRD(3) is set to the number of rows inserted.

Dependant table insertion: (Table with foreign Key)

Each non-null value you insert into a foreign key column must be equal to some value in the primary key (the primary key is in the parent table). If any field in the foreign key is null, the entire foreign key is considered null. If you drop the index enforcing the primary key of the parent table, an INSERT into either the parent table or dependent table fails.

DML-UPDATE

UPDATE statement is used to modify the data in a table. The SET clause names the columns you want to update and provide the values you want them changed to. The condition in the WHERE clause locates the row(s) to be updated. If you omit the WHERE clause, DB2 updates every row in the table or view with the values you supply.

If DB2 finds an error while executing your UPDATE statement (for instance, an update value that is too large for the column), it stops updating and returns error codes in the SQLCODE and SQLSTATE host variables and related fields in the SQLCA. No rows in the table change (rows already changed, if any, are restored to their previous values). If the UPDATE statement is successful, SQLERRD(3) is set to the number of rows updated.

Syntax:

```
UPDATE table-name SET column1 =value1, column2 =value2 [WHERE condition];
```

DML-DELETE

DELETE statement is used to remove entire rows from a table. The DELETE statement removes zero or more rows of a table, depending on how many rows satisfy the search condition you specified in the WHERE clause. If you omit a WHERE clause from a DELETE statement, DB2 removes all the rows from the table or view you have named. The DELETE statement does not remove specific columns from the row. SQLERRD(3) in the SQLCA contains the number of deleted rows.
Syntax: DELETE FROM table-name [WHERE CONDITION]

DML – NULLS

One of the 12 CODD Rules for relation database system is ' NULL values are supported for representing missing information in a systematic way irrespective of the data type'. DB2 supports NULL values.

NULL IN SELECT STATEMENT

NULL is defined as Unknown value in RDBMS terminology.

Two unknown values cannot be same. As EQUAL can be used only for known values, COLUMN1=NULL is meaningless. In the where predicate if NULL needs to be checked, WHERE COLUMN1 IS NULL is the right way of coding. If GROUPBY is done on a NULL column, then all the columns whose value is unknown (NULL) forms one GROUP.

Example, If QUALIFICATION is a column defined with NULL attribute in EMPTABLE, then SQL for retrieving all the employees whose QUALIFICATION is not yet known is:

```
SELECT EMPNAME,QUALIFICATION FROM EMPTABLE
  WHERE QUALIFICATION IS NULL.
```

High-level languages don't have any NULL concept. So if the column is NULL, then the host variable corresponds to that column should be set to zero (numeric) or spaces (non-numeric). This can be done in the programming. But for doing this we should know whether the column is NULL or NOT. NULL indicators are used for this purpose. NULL indicator is a 2 byte field (S9(4) COMP). Negative value,–1 in NULL indicator field indicate that the column is NULL.

```
EXEC SQL
SELECT QUALIFICATION
   INTO :WS-QUALIFICATION :WS-QUALIFICATION-NULL
   FROM EMPTABLE
   WHERE EMPID=2052
END-EXEC
IF SQLCODE = 0
   PERFORM 1000-CHECK-FOR-NULLS
….
100-CHECK-FOR-NULLS.
      IF WS-QUALIFICATION-NULL < 0 THEN
         MOVE SPACES TO WS-QUALIFICATION
      END-IF.
```

If QUALIFICATION of EMPID = 2052 is NULL and if you didn't code null indicator, then the query will fail with SQLCODE –305 and the error message corresponding to this code is 'THE NULL VALUE CANNOT BE ASSIGNED TO OUTPUT HOST VARIABLE NUMER posit-num BECAUSE NO INDICATOR VARIABLE IS SPECIFIED '

This kind of NULL check (100-CHECK-FOR-NULLS) is highly recommended for numeric fields. Otherwise the program may abnormally end at some place down the line, when the field is used for some arithmetic operation.

Instead of NULL indicator, VALUE function can also be used.

```
SELECT VALUE(QUALIFICATION,' ')
   INTO :WS-QUALIFICATION
   FROM EMPTABLE
  WHERE EMPID=2052
```

If QUALIFICATION is NULL, then ` ` will be moved to WS-QUALIFICATION.

NULL IN INSERT/UPDATE STATEMENT

The concept is same. DB2 informs the NULL presence thru' the NULL indicator to the programming language. In similar way, the programming language should inform the NULL to DB2 by NULL indicator.

Just before INSERT or UPDATE move –1 to NULL indicator variable and use this variable along with host variable for the column to be inserted/updated. Once –1 is moved to null indicator, then independent of value presence in the host variable, NULL will be loaded. In the following query, though 'B.E' is moved to host variable, it will not be loaded into the table as null indicator has –1. It should have been set to 0 before loading.

```
MOVE –1 TO WS-QUALIFICATION-NULL.
MOVE 'B.E' TO WS-QUALIFICATION.
EXEC SQL
    UPDATE EMPTABLE
    SET QUALIFICATION = :WS-QUALIFICATION :WS-QUALIFICATION-NULL
    WHERE EMPID=2052
END-EXEC.
```

It is common feeling that NULL provides more trouble than benefit. So it is always better to specify NOT NULL or NOT NULL WITH DEFAULT for all the columns.

Values in null indicator

A negative value in null indicator field implies that the column is NULL and a positive value or ZERO in null indicator implies that the column is NOT NULL. Value of –2 in null indicator indicates that the column has been set to null as a result of a data conversion error.

DCL-GRANT and REVOKE commands

Data Control Language consists of commands that control the user access to the database objects. The Database Administrator (DBA) has the power to give (GRANT) and take (REVOKE) privileges to a specific user, thus giving or denying access to the data.

Syntax:

```
GRANT access ON object TO USER-ID | PUBLIC [WITH GRANT OPTION].
REVOKE access ON object FROM USER-ID | PUBLIC.
```

Example:

```
GRANT SELECT, UPDATE ON EMPTABLE TO SMSXL86.
GRANT ALL ON EMPTABLE TO SMSXL86.
REVOKE CREATE TABLE, CREATE VIEW FROM SMSXL86.
```

REVOKE INSERT ON EMTABLE FROM SMSXL86.

COMMIT and ROLLBACK

A transaction, or a unit of work, is a recoverable sequence of one or more SQL operations that are grouped together as a single unit, usually within in an application process.

All the changes made to the database since the initiation of transaction, are made permanent by COMMIT. ROLLBACK brings back the database to the last committed point.

Syntax:
EXEC SQL COMMIT END-EXEC
EXEC SQL ROLLBACK END-EXEC

DB2 Restart Logic:

Usually there is only one COMMIT or ROLLBACK just before the termination of the transaction. But it is not preferred always.

If the program is updating one million records in the table space and it abnormally ends after processing ninety thousand records, then the issued 'ROLLBACK' brought the database back to the point of transaction initiation. The restart of the program must have to process ninety thousand successful-but-not-committed updates once again. The repeated cost occurred is due to Bad design of the application.

If the program is expected to do huge updates, then commit frequency has to be chosen properly. Let us say, after careful analysis, we have designed our application COMMIT frequency as thousand records. If the program abnormally ends while processing 1500^{th} record, then the restart should not start from first record but from 1001^{st} record. This is done using restart logic.

Create one temporary table called RESTARTS with a dummy record and inserts one record into the table for every commit with key and occurrence of commit. This insertion should happen, just BEFORE the issue of COMMIT.

First paragraph of the procedure should read the last record of the table and skipped the records that are already processed and committed (1000 in the previous case). After the processing of all the records (one million), delete the entries in the table and issue one final COMMIT.

JOINS

Most of the times, the complete information about an entity is retrieved from more than one table. Retrieving information from more than one table can be done in two ways. One method is JOIN and the other method is UNION. We have already discussed about UNION. It is used to get information about more entities. In other words, it returns more rows. JOIN is used to get detail information about entities. In other words, it returns more columns.

There are two kinds of JOIN. Inner join returns rows from two or more tables based on matching values. Outer join returns rows from tow or more tables independent of matching or non-matching.

JOIN: INNER

Two or more tables are joined together using the column having equal values among them.

EMPTABLE

EMP_ID	EMP_NAME	DESIG
100	MUTHU	TL
101	DEVI	SSE

SALTABLE

DESIG	SALARY
SSE	400000
SE	300000

```
SELECT EMP-NAME SALARY FROM EMPTABLE, SALTABLE WHERE
        EMPTABLE.DESIGNATION = SALTABLE.DESIGNATION
```

It can also be coded as

```
SELECT EMP-NAME SALARY FROM EMPTABLE INNER JOIN SALTABLE ON
        EMPTABLE.DESIGNATION = SALTABLE.DESIGNATION
```

Result

DEVI 400000

Since there is no TL designation in the SALTABLE, MUTHU will not appear in the output.

JOIN:FULL OUTER JOIN

The clause FULL OUTER JOIN includes unmatched rows from both tables. Missing values in a row of the result table contain nulls.

```
SELECT EMP-NAME SALARY FROM EMPTABLE FULL OUTER JOIN SALTABLE ON
        EMPTABLE.DESIGNATION = SALTABLE.DESIGNATION
```

Result:

```
MUTHU  --
DEVI   400000
---    300000
```

JOIN:LEFT OUTER JOIN

The clause LEFT OUTER JOIN includes rows from the table named before it where the values in the joined columns are not matched by values in the joined columns of the table named after it.

```
SELECT EMP-NAME SALARY FROM EMPTABLE LEFT OUTER JOIN SALTABLE ON
        EMPTABLE.DESIGNATION = SALTABLE.DESIGNATION
Result
MUTHU   --
DEVI     400000
```

JOIN:RIGHT OUTER JOIN

The clause RIGHT OUTER JOIN includes rows from the table named after it where the values in the joined columns are not matched by values in the joined columns of the table named before it.

```
SELECT EMP-NAME SALARY FROM EMPTABLE RIGHT OUTER JOIN SALTABLE ON
        EMPTABLE.DESIGNATION = SALTABLE.DESIGNATION

Result
DEVI     400000
----     300000
```

Restrictions on Outer Joins

1. VALUE or COALESCE function can be used only with FULL OUTER join.
2. Only '=' can be used as comparison operator in FULL OUTER join.
3. Multiple conditions can be coded with AND. 'OR' and 'NOT' are not allowed.

Sub-queries

Sub-Queries are nested SELECT statements. Sub-query enables a user to base the search criteria of one SELECT on the result of another SELECT statement. It is always part of a predicate.

Syntax: (MAIN-QUERY) WHERE operand operator (SUB-QUERY)

Sub-queries can appear in the predicates of other sub-query. This is called nested sub-query and the nesting is possible up to 15 levels.

Usually sub-query executes only once and it is executed first and the value returned by sub-query will be used for main query. This kind of sub-query is called as un-correlated sub-query. When the sub-query refers to the table referred in the main-query then the sub-query is executed for every execution of main-query and this kind of query is called correlated sub-query. Correlated sub-queries cannot be used for update/insert/delete purpose.

Example:

The following query lists the employees who get more than the average salary of their department.

```
SELECT EMPNAME,DEPTNUM,SALARY
   FROM EMPTABLE
 WHERE SALARY > (SELECT AVG(SALARY)
                   FROM DEPTNUM
                  WHERE EMPTABLE.DEPTNUM=DEPT.DEPTNUM)
```

OPTIMIZER- ACCESS PATH SELECTION

Generally, the fastest way to access DB2 data is with index. Indexes are structured in such a way to increase the efficiency of finding a particular piece of data. DB2 uses many different algorithms to traverse an index structure. These algorithms are designed to elicit optimum performance in a wide variety of data access scenarios.

To use an index:

1. One of the predicates for the SQL statement must be index-able.
2. One of the columns (in any index-able predicate) must exist as column in an available index.

There are various types of indexes are available. They are:

1. Direct Index Look-Up.
2. Index Scan – Matching index scan.
3. Index Scan – Non-matching index scan.
4. Index-Only access.
5. List pre-fetch
6. Multi-index access (AND & OR)

Example: The primary key of the table TASKTAB is a composite key with columns CLIENT_ID, PROJECT_ID, MODULE_ID.

Direct Index Look-Up:

Values are provided for each column in the index. In the above example, if the value for all the three columns are given in the where clause, then there will be a direct index look up.

```
SELECT MODULE_ID, MODULE_STATUS FROM TASKTAB
WHERE CLIENT_ID = 100 AND PROJECT_ID=10 AND MODULE_ID = 1
```

If only CLIENT_ID and MODULE_ID is given then direct index look-up is not possible. Index scan would occur.

Matching Index Scan or Absolute Positioning

It starts with root page and work down to a leaf page in much the same manner as direct index loop up does. However since the full key is unavailable, DB2 must scan the leaf pages using the values it does have, until all matching values have been retrieved. As it starts at root page, the value of first column(s) should be given.

```
SELECT MODULE_ID, MODULE_STATUS FROM TASKTAB
WHERE CLIENT_ID = 100 AND PROJECT_ID=10
```

The query use matching index scan and returns all the module id and module status for the given client and project.

Non-matching index scan or Relative positioning

Non-matching index begins with the first leaf page in the index and sequentially scans subsequent leaf pages applying the available predicates. If any ORDER BY or GROUP BY is coded, then optimizer prefers non-matching index scan than table space scan.

```
SELECT CLIENT_ID, MODULE_ID, MODULE_STATUS FROM TASKTAB
WHERE PROJECT_ID=10
```

ORDER BY MODULE_ID

Index-only access

If all the data required by the query is available in index, then DB2 can avoid reading the data page and perform non-matching index scan.
This special type of non-matching index scan is called index-only access.
SELECT CLIENT_ID FROM TASKTAB WHERE PROJECT_ID = 10 AND MODULE_ID=1

Sequential pre-fetch and List pre-fetch

Sequential pre-fetch is a technique used by optimizer during SEQUENTIAL table space SCAN. It can be though of as a read ahead mechanism invoked to pre-fill DB2 buffers so that data is already in memory before it is requested.

List Pre-fetch is a technique used by optimizer during INDEXED access. If the cluster ratio of the index is poor (less than 80%) then RIDs of matching indexes are sorted based on data page number and then the data-page is accessed.

This significantly reduces the I-O as same page will not be read second time in this method. Clustered index access doesn't use this technique as already the data rows are in the same order of index. (Cluster ratio is 1)

Multi index access

When more than two indexes are used in where clause..
1.Using the first index, RIDs matching the requirement are chosen and sorted based on data-page number.
2.Using the second index, RIDs matching the requirement are chosen and sorted based on data-page number.
3.If the indexes are joined using OR, then the union of RID list of 1 and 2 is used to access data page.
4. If the indexes are joined using AND, then the intersection of RID list of 1 and 2 is used to access data page.

Table Space Scan

If there is no index-able column in the predicate or DB2 decides not to use available index, then the optimizer choose table space scanning.

Table space scanning is like sequential read of file. All the rows of all the pages are read sequentially and the rows satisfying the selection criteria in the where clause is chosen for further processing.

Query Parallelism

This is another technique that can be applied by the optimizer is query parallelism. When parallelism is invoked, an access path can be broken up into parallel groups. Each parallel group represents a series of concurrent operations with the same degree of parallelism. Degree of parallelism refers to the number of concurrent tasks used to satisfy the query.

Query I/O parallelism (as of DB2 V3):

Breaks the data access for the query into concurrent I/O streams executed in parallel should reduce the overall elapsed time for the query. With query I/O parallelism, DB2 is limited to operate on a single processor for each query.

Query CP Parallelism (as of DB2 V4):

The large query is decomposed into multiple smaller queries that can be executed concurrently with one another on multiple processors. CP parallelism always uses I/O parallelism. This should further reduce the elapsed time of the query.

Query Sysplex Parallelism (as of DB2 V5):

It further enhances parallel operations by enabling a single query to be broken up and run across multiple DB2 subsystems within a data-sharing group. It takes the advantage of the processing power of multiple DB2 subsystems.

Filter factor

Filter factor is a ratio that estimates I/O costs. The optimizer calculates the filter factor for a query's predicates based on the number of rows that will be filtered out by the predicates. That is, if the predicate is most restrictive, the filter factor would be low, the I-O cost would be less and the query will be more efficient.
For an indexed column, FIRSTKEYCARDF column of SYSIBM.SYSINDEXES consists of number of distinct values in the column.

If the predicate is column=value then filterfactor = 1/FIRSTKEYCARDF
If the predicate is column <> value then filterfactor = (1-1/ FIRSTKEYCARDF).
This case involves more I-O is involved and inefficient.

EXPLAIN

We have already seen that DB2 optimizer chooses the access path for the query. If we want to know what access path DB2 chose during the plan preparation, then the request should be placed to DB2. The request can be done two ways:

Method 1: Use EXPLAIN (YES) parameter in the BIND card.
SELECT * FROM ownerid.PLAN_TABLE ORDER BY APPLNAME, COLLID, VERSION, PROGNAME, TIMESTAMP DESC, QUERYNO, QBLOCKNO, PLANNO

Method 2: Use the following command directly in the program or in SPUFI or QMF.
Syntax: EXPLAIN ALL SET QUERYNO=integer FOR SQL-statement.

Before posting the request, there should be a PLAN_TABLE under the user-id, based on model SYSIBM.PLAN_TABLE. PLAN_TABLE is a standard table that must be defined with predetermined columns, data types and lengths. During bind process, DB2 optimizer briefs the access path chose by it in the PLAN_TABLE.

If you want to query the access path for single query then use the query below:
SELECT * FROM PLAN_TABLE WHERE QUERYNO=integer OREDERBY QBLOCKNO, PLANNO.

Now the access path taken by the optimizer is loaded into PLAN_TABLE. One should know the meaning of PLAN_TABLE columns and values to understand the access path. DB2V5 has 46 columns in PLAN_TABLE.

PLAN_TABLE Column name	Meaning
QUERYNO	Integer value assigned by the user issuing the EXPLAIN or DB2
QBLOCKNO	Integer identifying level of sub-query or union in a SQL statement. The first sub-select is numbered as 1, the second as 2 and so on.
APPLNAME	Name of the PLAN or PACKAGE based on BIND stage of static SQL. It will be spaces for SPUFI request.
PROGNAME	Name of the program in which SQL statement is embedded. DSQIESQL for SPUFI EXPLAIN

PLANNO	Integer value saying the step of the plan in which QBLOCKNO is processed. (The order in which plan steps are undertaken)
METHOD	Integer value identifying the access method used for the given step. 1- Nested loop join 2-Merge scan join 3-independent sort as a result of ORDERBY, GROUP BY, DISTINC or IN 4-Hybrid Join.
ACCESSTYPE	Technique used for access. I – Indexed access I1 – One fetch index scan R – Table Space Scan N – Index access with In predicate M – Multiple index scan MX – specification of the index name for multiple index access. MI – Multiple index access by RID intersection MU – Multiple index access by RID union
MATCHCOLS	Number of index columns used in indexed access. Access type should be I, I1, M, MX.
INDEXONLY	'Y' indicates access to the index is sufficient to satisfy the query.
PREFETCH	Indicates which type of pre fetch will be used. S–Sequential, L- List, Blank–Unknown at bind time.
PARALLELISM MODE	Indicates type of parallelism used at bind time. I – I/O Parallelism, C – CPU Parallelism X – Sysplex Parallelism, Blank – No or determined at run time.
QBLOCK_TYPE	Type of SQL operation performed for the query block. CORSUB – Correlated sub query NCOSUB- Non correlated sub query UPDCUR, DELCUR – Update and Delete using WHERE CURRENT OF option of cursor. SELECT, INSERT, UPDATE, DELETE – Type of DML. SELUPD – SELECT with FOR UPDATE OF.
JOIN_TYPE	Indicates type of join implemented. F – Full Outer Join L – Left Outer Join (or converted right outer join) Blank – Inner join or no join.
TIMESTAMP	Date and time the EXPLAIN for this row was issued.

Other columns are:
CREATOR, TNAME, TABNO, ACCESSCREATOR, ACCESSNAME, SORTN_UNIQ, SORTN_JOIN, SORTN_ORDERBY, SORTM_GROUPBY, SORTC_UNIQ, SORTC_JOIN, SORTC_GROUPBY, SORTC_GROUP, TSLOCKMODE, REMARKS, COLUMN_FN_EVAL, MIXOPSEQ, VERSION, COLLID, ACCESS_DEGREE, ACCESS_PGROUP_ID, JOIN_DEGREE, JOIN_PGROUP_ID, SORTC_PGROUP,ID, SORTN_PGROUP_ID, MERGE_JOIN_COLS, CORRELATION_NAME, PAGE_RANGE, GROUP_MEMBER, IBM_SERVICE_DATA,WHEN_OPTIMIZE, BIND_TIME.

DYNAMIC SQL

Static SQL is embedded in application program and the only values that can change are the values of the host variables in the predicates. Dynamic SQL are characterized by the capability to change columns, tables and predicates during a program's execution.

Advantages: Flexibility and best access path (as the bind happens at the run time using latest RUNSTATS information)

Disadvantage: Slow as the runtime is bind time + execution time.

Types of Dynamic SQL

1.EXECUTE IMMEDIATE:

Move the SQL statement to be executed into the host variable and issue execute immediate command.

```
 01 WS-HOST.
      49 WS-HOST-LENGTH PIC S9(04) COMP.
      49 WS-HOST-VAR       PIC X(40).
   MOVE +40 TO WS-HOST-LENGTH
   MOVE  "SET EMP_NAME = 'MUTHU' WHERE EMP_NO=2052"   TO WS-HOST-TEXT.
   EXEC SQL EXECUTE IMMEDIATE: WS-HOST-VAR END-EXEC.
```

Disadvantages:
1.It cannot be used for SELECT.
2.Executable form of the SQL will be deleted once it is executed.

2.EXECUTE WITH PREPARE:

This is equivalent to first form but here the SQL is prepared before execution and so the second disadvantage is avoided in this method.

Form: 1

```
   MOVE +40 TO WS-HOST-LENGTH
   MOVE  "SET EMP_NAME = 'MUTHU' WHERE EMP_NO=2052"   TO WS-HOST-TEXT.
   EXEC SQL PREPARE RUNFORM FROM :WS-HOST END-EXEC.
   EXEC SQL EXECUTE RUNFORM END-EXEC.
```

Form: 2

Parameter markers can be used in place of constant values. This acts like placeholder for the host variables in the SQL.

```
   MOVE +40 TO WS-HOST-LENGTH
   MOVE  "SET EMP_NAME = 'MUTHU' WHERE EMP_NO=?"       TO WS-HOST-TEXT.
   EXEC SQL PREPARE RUNFORM FROM :WS-HOST END-EXEC.
   MOVE 2052 TO WS-EMP-NUM
   EXEC SQL EXECUTE RUNFORM USING :WS-EMP-NUM END-EXEC.
```

Disadvantage:
Select statement is not supported.

3.SELECT with STATIC Columns (Fixed-list SELECT)

The following method is used for data retrieval but the column to be selected are known and unchanging.
Let the query be ' SELECT EMP_NAME FROM EMPTABLE WHERE DEPT_NO=?'

```
MOVE 'Select SQL ' to WS-HOST.
EXEC SQL DECLARE CSR1 CURSOR FOR SELSQL END-EXEC
EXEC SQL PREPARE FROM :WS-HOST END-EXEC
MOVE 100 TO WS-DEPT-NO
EXEC SQL OPEN CSR1 USING :WS-EMP-NO END-EXEC
PERFORM UNTIL SQLCODE =100
  EXEC SQL FETCH CSR1 INTO :WS-EMPNAME END-EXEC
END-PERFORM.
EXEC SQL CLOSE CSR1 END-EXEC.
```

Disadvantage:
1.Table name cannot be changed. Number of columns cannot be modified during run time.

The fourth type of dynamic SQL known as 'varying-list select' provide hundred percent flexibility where the number of columns and even the table we are accessing can be changed during run time. This uses the copybook SQLDA for its purpose.

INTEGRITY

Integrity means the accuracy, correctness or validity of data contained in the database. Maintaining integrity is not an easy task in a multi user environment. So the task of maintaining integrity is handled by the system than the user.

Domain Integrity.

It ensures that a value in a column is a member of column's domain or legal set of values. Simple example is, alphabetic data on the integer column is restricted.

Entity Integrity.

It means every row in a table is unique. To ensure entity integrity, developer has to define a set of column(s) as primary key.

Referential Integrity.

It ensures the data integrity between the tables related by Primary (Parent) and Foreign (Child) keys.

Concurrency and its issues:

DB2 allows more than one application to access the same data at essentially the same time. This is known as concurrency. Concurrency results the following problems.

1.The LOST UPDATE Problem.

Time	Transaction-A	Transaction-B
T0	Read row R1.	
T1		Read row R1.
T2	Update row R1.	
T3		Update row R1.

Transaction-B overwrites the change done by Transaction-a at T2.
In other words, the update of transaction-A is lost.

2.Uncommitted Dependency Problem

It arises when a transaction is allowed to retrieve (update) a row that has been updated by another transaction and has not yet been committed by the other transaction.

Time	Transaction-A	Transaction-B
T0	Update row R1.	
T1		Read and Update row R1.
T2	Rollback	

Transaction-B updates row R1 based on the uncommitted change done by Transaction-A. Transaction-B rolled back its changes.

3.Data inconsistency.

Let us say there is 2 account A1,A2 and account A3 should have sum of amounts in A1 and A2. Say A1 = 1000 A2 = 2000 A3=3000

Time	Transaction-A	Transaction-B
T0	Read row A1.(1000)	
T1		Read row A1 and update to 2000.
T2	Update row A2.(3000)	
T3	Calculate A3 and update. (4000)	
T4	Commit	Commit

Now the value of A1=2000, A2=3000 but A3=4000 == > Data inconsistent as A3 is expected to be 5000 as per design.

Locking

Locking solves the concurrency issues described above. Locking prevent another transaction to access the data that is being changed by the current transaction. Three main attributes of lock are mode, size and duration.

Mode of the lock:

It specifies what access to the locked object is permitted to the lock owner and to any concurrent application process.

Exclusive (X) – The lock owner can read or change the locked page. Concurrent processes cannot acquire ANY lock on the page nor they access the locked page.

Update (U) – The lock owner can read but cannot change the locked page. However the owner can promote the lock to 'X' and update. Processes concurrent with the U lock can acquire 'S' lock and read the page but another 'U' lock is not possible.

Share (S) - The lock owner and any concurrent processes can read but not change the locked page. Concurrent processes can acquire S or U locks on the page.

The above three are with respect to pages. Three more kind of locks are possible at table /table space level. They are: Intent Share (IS), Intent Exclusive (IX) and Share/Intent Exclusive (SIX)

SIZE of the lock

It can be ROW, PAGE, TABLE and TABLESIZE.

Duration of the lock

It is the length of time the lock is held. It varies according to when the lock is acquired and released. This is controlled by ACQUIRE, RELEASE and ISOLATION LEVEL parameters of BIND.

LOCK Issues

Locking is introduced to avoid concurrency issues. Locking resulted suspension, timeout and deadlock problems. But LOCK is needed for data integrity.

Suspension.

IRLM suspends an application process if it requests a lock on an object that is already owned by another application process and cannot be shared. The suspended process is said to be in 'Wait Stage' for the lock. The process resumes when the lock is available.

Timeout.

When the object is in wait stage more than predefined time, then the process (program) is terminated by DB2 with SQL return code –911 or –913 meant for TIMEOUT. The IRLMRWT of DSNZPARM determines the duration of the wait time.

Deadlock.

When two or more transactions are in simultaneous wait stage, each waiting for one of others to release a lock before it can proceed, DEADLOCK occurs.

If deadlock occurs, transaction manager of DB2 breaks it by terminating one of the transactions and allows the other to process. Later, the application programmer will restart the terminated transaction.

Lock and Latch

A true lock is handled by DB2 using an external component called IRLM. However, when it is practical, DB2 tries to lock pages without going to IRLM. This type of lock is known as LATCH. Latches are internally set by DB2 without going to IRLM.

Initially latches are used to lock only DB2 index pages and internal DB2 resources. DB2 V3 uses latching more frequently for data pages also. Latches are preferred when a resource serialization is required for a short time. Locks and Latches guarantee data integrity at the same level.

Advantage of Latch over Lock:
1.Cross-memory service calls to IRLM is eliminated.
2.Latch requires about one third the number of instructions as a lock.

Lock Promotion

When lock is promoted from lower level to upper level, it is called lock promotion. DB2 acquire 'U' lock to update a row. But at the same concurrent processes can access the row in 'S' lock. During update, it needs 'X' lock and to get the 'X' lock, it waits for concurrent processes to release 'S' locks. It promotes the current lock from 'U' to 'X' and this is called LOCK PROMOTION ON MODE. ('U' – Update in Pending and 'X' – Update in Progress)

When the number of page locks exceeds predefined limit, it is promoted to table or table space level and this is LOCK PROMOTION ON SIZE.

Lock Strategy

DB2 decides the lock to be applied based on the following:

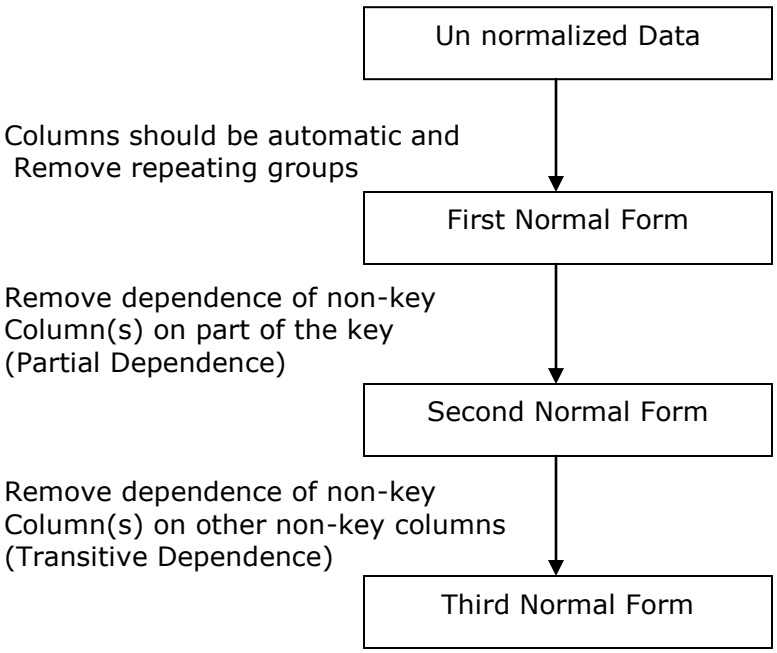
1. Lock size declared at the time of table space creation.
2. Type of SQL statement – SELECT or UPDATE
3. Explicit LOCK statement in the program.
4. ACQUIRE and RELEASE option chosen by user at the BIND time.
5. Isolation level specified at bind time.
6. The NUMLKTS and NULLKUS limit.

Normalization

Data Normalization describes the process of designing a database and organizing data to take best advantage of relational database principles. It is the process of putting one fact in one appropriate place. This optimizes the updates at the expense of retrievals.

When a fact is stored in only one place, retrieving many different but related facts usually requires going to many different places. This tends to slow the retrieval process but update is easy and faster as the fact you are updating exists in only one place.

The process involves five levels. Tables are usually normalized till level three.



The three levels can be memorized using the statement below:

" The values in a row are dependant on the key, the whole key, and nothing but the key, so help me CODD"

In the fourth level, multi-valued dependencies are removed and in the fifth level, remaining anomalies are removed.

De-normalization

De-normalization is the process of putting one fact in more than one place. It is the reverse process of normalization. This will speed up the retrieval process at the expense of data modification. De-normalization is not a bad decision when a completely normalized design is not performing optimally.

System Catalog Tables

DB2 catalog is contained in a single database (DSNDB06). The 54 tables in the DB2 catalog collectively describe the objects and resources available to DB2.

Prior to Version5, there was an additional database called Communication Database (CDB) was used for establishing and documenting distributed DB2 connections. In the version 5, it has been renamed and rolled into DB2 catalog.

Each and every table has its own importance. So explaining all the tables and the columns in it is highly impossible. Refer the query section to know about some of the frequently used queries on SYSIBM tables.

1. SYSIBM.SYSDUMMY1 is a dummy table. When you want to do some arithmetic operation or retrieving system information like current-date, you can query this table.
   SELECT 10.25*20.25 FROM SYSIBM.SYSDUMMY1;

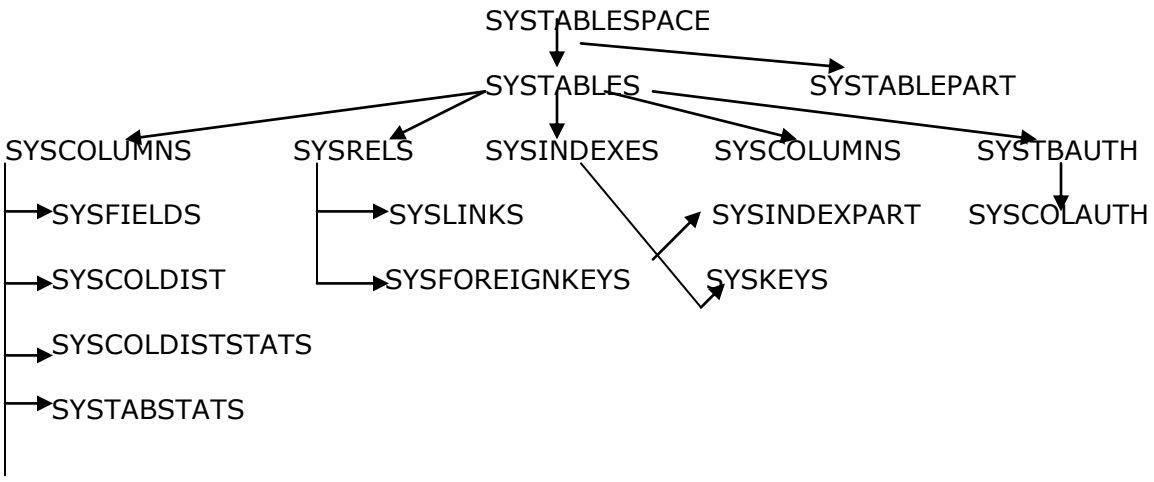
2. When a DBRM is bound to PLAN, all the SQL statements are placed into SYSSTMTDB2 table. When a DBRM is bound to PACKAGE, all of its SQL statements are placed into SYSPACKSTMT table. Information about the DBRMS that were bound to PLAN or PACKAGE, are stored in SYSDBRM table. DBRM just created but not yet bound is not registered anywhere. (Remember, pre-compiler don't use DB2 at all and DBRM is an output of pre-compiler)

3. Information about plan is stored in SYSDBRM, SYSPACKAUTH, SYSPACKLIST, SYSPLAN, SYSPLANDEP, SYSPLANAUTH, SYSPLSYSTEM, SYSSTMT and SYSTABAUTH.

4. Information about the package is stored in SYSPACKAGE, SYSPACKAUTH, SYSPACKDEP, SYSPLSYSTEM, SYSSTMT and SYSTBAUTH.

5. DB2 Catalog stores only information about the PLAN. The executable form of PLAN, called Skeleton cursor table (SKCT), is stored in the DB2 directory in the SYSIBM.SCT02 table.

6. Information about Table space/ Table and Indexes are stored in the following Tables.



⟶SYSINEXSTATS

Triggers (Available in version 6)

Trigger is a piece of code that is executed in response to a data modification statement. (insert/update/delete). To be a bit more precise: triggers are event-driven specialized procedures that are stored in, and managed by the RDBMS. Each trigger is attached to a single, specified table. Triggers can be thought of as an advanced form of "rule" or "constraint" written using an extended form of SQL. Therefore triggers are automatic, implicit, and non-bypass able.

Nested Triggers

A trigger can also contain insert, update, and delete logic within itself. Therefore, a trigger is fired by a data modification, but can also cause another data modification, thereby firing yet another trigger. When a trigger contains insert, update, and/or delete logic, the trigger is said to be a nested trigger. Most DBMSs, however, place a limit on the number of nested triggers that can be executed within a single firing event. If this were not done, it could be quite possible to having triggers firing triggers ad infinitum until all of the data was removed from an entire database!

Triggers cannot be attached to the following tables:
A System Catalog Table, PLAN_TABLE, STATEMENT_TABLE, DSN_FUNCTION_TABLE
View, Alias, Synonym, Any table with a three-part name

Trigger in place of RI

Triggers can be coded, in lieu of declarative RI, to support ALL of the RI rules. We can specify only ON DELETE rules in the DDL statement. UPDATE and INSERT rules of RI can be implemented using triggers.

Sample Trigger:

```
CREATE TRIGGER SALARY_UPDATE
  BEFORE UPDATE OF SALARY
  ON EMP
  FOR EACH ROW MODE DB2SQL

 WHEN (NEW.SALARY > (OLD.SALARY * 1.5))
 BEGIN ATOMIC
  SIGNAL SQLSTATE '75001' ('Raise exceeds 50%');
 END;
```

The trigger executes once for each row. So if multiple rows are modified by a single update, the trigger will run multiple times, once for each row modified. Also, the trigger will be run BEFORE the actual modification occurs. Finally, take special notice how NEW and OLD are used to check values before and after the update.

Triggers Vs Stored Procedures

Triggers are also similar to stored procedures. Both consist of procedural logic that is stored at the database level. However, stored procedures are not event-driven and are not attached to a specific table. A stored procedure is explicitly executed by invoking a CALL to the procedure (instead of implicitly being executed like triggers). Additionally, a stored procedure can access many tables without being specifically associated to any of them. DB2 has supported stored procedures since Version 4.

DB2 Utilities

Most of the time, application programmer doesn't need any knowledge on DB2 utilities. Backup, Reorganization, Recovery and maintenance of database are taken care by the DBA of the project.

DB2 utilities are briefed here to just familiar with it.

DSNUPROC is the DB2 catalogued procedure that executes the program DSNUTILB. DSNUTILB is a master program and it calls other programs based on the first word of control card. The first word of control card should be utility that needs to be invoked.

Most of the DB2 Utility programs are divided into four major categories.

Utility	Purpose
COPY (Backup and Recovery)	It is used to create image copy backup dataset for a complete table space or a single partition of a table space. It can be of type FULL or incremental. Based on number of modified pages after the previous backup, prefer FULL or incremental image copy. Successful copy details are loaded in SYSIBM.SYSCOPY.
MERGECOPY (Backup and Recovery)	It is used to create full image copy from incremental image copies.
QUIESCE (Backup and Recovery)	It is used to record a point of consistency for related application or system tables. Usually done before image copy.
RECOVER (Backup and Recovery)	It is used to restore DB2 table space and indexes to a specific point in time. It uses image copy and DB2 logs information to roll back the table space content to the old one. RECOVER INDEX generated new index data from current table space data. It is actually regeneration than restoring. So it is followed by RECOVERY TABLESPACE, which is restoring.
LOAD (Data Organization)	It is used to accomplish bulk inserts to DB2 tables. It can add rows to a table retaining the current data or it can replace the existing data with new data.
REORG (Data Organization)	It is used to re-clustering the data in table space and resetting the free space to the amount specified in the CREATE DDL and deletes and redefining VSAM datasets for STOGROUP defined objects.
CHECK, REPAIR, REPORT, DIAGNOSE	These are data consistency utilities. They are used to monitor, control and administer data consistency errors.
RUNSTATS (Catalog Manipulation Utility)	It collects statistical information for DB2 tables, table spaces, partitions, indexes and columns. It places this information into DB2 catalog tables. The DB2 optimizer uses the data in these tables to determine optimal access path for the SQL queries. CATMAINT, MODIFY and STOSPACE are the other catalog manipulation utilities.

How the load card of LOAD utility looks?

```
LOAD DATA INDDN(SYSREC) LOG(NO) NOCOPYPEND
RESUME YES                          => Resumes all the records from first to last.
ENFORCE CONSTRAINTS
INTO TABLE DB1.EMPTABLE
(EMPNAME POSITION(1) CHAR(20),
 EMPADDRESS POSITION(25) VARCHAR
 EMPPROJECT POSITION(50) CHAR(3))
```

RESUME YES => Resumes all the records from first to last.
RESUME NO => Table should be empty.
REPLACE => It overrides the previous record set with current record set.

SYSREC dataset should contain

```
1               25                                  50
XXXX             XXXXXXXXXXXXXXXXXXX                 XXXXX
YYYYY            YYYYYYYYYYY                         YYYYY
```

LOAD Data statement describes the data to be loaded. In the above example, SYSREC is coded and it should be one of the DDNAME in JCL, which points to actual dataset.
Data are loaded in the order they appear in the dataset. Automatic data conversion between compatible data types is done if needed.

What are COPY PENDING and CHECK PENDING status?

COPY PENDING and CHECK PENDING status on the table space restricts any updates on table space.

If you LOAD or REORG the table space with the option LOG NO, then the table space get into COPY PENDING status. The meaning is, an image copy is needed on the table space.

If you LOAD the table space with ENFORCE NO option, then the table space get into CHECK PENDING status. The meaning is table space is loaded without enforcing constraints. CHECK utility needs to be run on the table space.

COPY Utility

```
COPY TABLESPACE EDSDB.TRG007TS
FULL YES |NO                YES FOR FULL IMAGE and NO FOR INCREMENTAL
SHRLEVEL CHANGE
```

MERGECOPY Utility

```
MERGECOPY TABLESPACE EDSDB.TRG007TS
NEWCOPY (YES/ NO)
```

YES. Merging all the incremental copies with the available full image copy and creates a new image copy.
NO. Merging all the incremental copies and prepares a single incremental image copy.

REPAIR Utility

```
//DSNUPROC.SYSIN    *
REPAIR OBJECT LOG YES SET TABLESPACE NPCBTC00.TASSEM01 NOCOPYPEND
/*
```

This card repairs the table space NPCBTC00.TASSEM01 and clears the copy pending status on the table space.

RUNSTATS Utility

```
//SYSIN    DD *
  RUNSTATS TABLESPACE NPCBTC00.TASSEM01
     TABLE (NPCBT9TS.TASTEM01)
       COLUMN (NPA, TELNO, CLS_MO, TR_CRT_MO, TR_CRT_YY,
             SVC1_TYP, RPT1_TYP, SRC1_TYP, CLR_MO, CLR_DY)
     INDEX (ALL)
     SHRLEVEL CHANGE
     REPORT YES UPDATE ALL
/*
```

REBUILD/RECOVER/REORG and CHECK DATA Utility

```
REBUILD INDEX (ALL) TABLESPACE(NPCBT9DB.NPCS7011)
 SORTKEYS

RECOVER TABLESPACE NPCTT9DB.NPCS9042

REORG TABLESPACE NPCTT9DB.NPCS7048
         UNLDDN SYSRE
         SORTDEVT SYSDA     SORTNUM    4
         WORKDDN(SYSUT1,SRTOUT)

 CHECK DATA TABLESPACE NPCTT9DB.NPCS7013
          SORTDEVT SYSDA     SORTNUM    4
          WORKDDN(SYSUT1,SRTOUT)
```

QUIESCE Card:

```
QUIESCE
   TABLESPACE NPCBT900.NPCS1002
   TABLESPACE NPCBT900.NPCS1004
```

START Database:

```
DSN SYSTEM(DB2X)
START  DATABASE(NPCBT900) SPACENAM(NPCS1001) ACCESS(RO)
START  DATABASE(NPCBT900) SPACENAM(NPCS1002) ACCESS(RO)
END
```

SQL – Guidelines for better performance

1. Pre-test all your queries in SPUFI or QMF before placing in program.
2. Avoid qualifiers for the tables in the program. Give it on the BIND card.
3. Unless really needed, avoid usage of DISTINCT, UNION and ORDER BY.
4. Avoid WHENEVER.
5. Code predicates on index-able columns – Code the most restrictive predicated first – Use equivalent data types in the predicates
6. Do not use arithmetic expressions in a predicate. – DB2 will not use index in this case.
7. Prefer BETWEEN in stead of '<= ' and '>='
8. Prefer 'IN' in stead of 'LIKE'
9. Avoid the usage NOT. (Except with NOT EXISTS)
10. Join using SQL instead of programming logic.
11. Use JOINS than Sub-query.
12. Minimize the number of tables in the JOIN.
13. When a SELECT statement is coded, use FOR FETCH ONLY. This enables block fetch option DB2 and leads to efficient retrieval.
14. If the cursor is used for update, use FOR UPDATE OF clause in DECLARE statement itself.
15. Though DECLARE CURSOR can be given in procedure division, code it in working storage section as a good practice. DECLARE cursor is not an executable statement and procedure division is for executable statements.
16. For existence checking, use constant. (..WHERE EXISTS (SELECT 1 FROM TAB1)
17. Give a thought on dropping indexes before large insertions. It avoids unorganized indexes and poor performance.
18. Don't USE Select * - If there is any change in DB2 table structure, then the program needs to be modified though it doesn't need the field.
    There is unnecessary I-O overhead.
19. Do the arithmetic operations while selecting the column(s) than doing in programming language

Famous SQL Queries

First and second maximum salary of EMPTABLE

```
SELECT MAX(SALARY) FROM EMPTABLE

SELECT MAX(SALARY) FROM EMPTABLE
  WHERE SALARY < (SELECT MAX(SALARY) FROM EMPTABLE)
```

Nth maximum salary from EMPTABLE

```
  SELECT SALARY FROM EMPTABLE A
   WHERE N = (SELECT COUNT(*)FROM EMPTABLE B
              WHERE B.SALARY >= A.SALARY)
```

How do you calculate day of the week?

```
SELECT DAYS(DATE(:WS-SQL-DT1))-(DAYS(DATE(:WS-SQL-DT1))/7) * 7
INTO :WS-DAY-OF-WEEK
FROM SYSIBM.SYSDUMMY1
```

0-6 means SUNDAY to SATURDAY respectively.

How do you get current date in DB2?

```
SET :WS-CURRENT-DATE  =  CURRENT DATE
```

Sample query to concatenate two columns

```
SELECT 'Mr.' ||FIRST_NAME||STF_LAST_NAME
  INTO  :WS-USER-NM
  FROM  CRS_STAFF
 WHERE  STF_CODE = :WS-I-USER-ID
```

How do I get the primary key of a table?

```
SELECT NAME
  FROM SYSIBM.SYSCOLUMNS
 WHERE TBCREATOR = <TABLE CREATOR NAME>
   AND TBNAME    = <TABLE NAME>
   AND KEYSEQ    > 0
```

Or

```
SELECT A.COLNAME,
       A.COLSEQ,
       A.ORDERING,
  FROM SYSIBM.SYSKEYS    A,
       SYSIBM.SYSINDEXES B
 WHERE A.IXNAME    = B.NAME
   AND A.IXCREATOR = B.CREATOR
   AND A.IXCREATOR = B.TBCREATOR
   AND B.TBNAME    = <TAB NAME>
   AND B.TBCREATOR = <TAB CREATOR>
```

```
  AND B.UNIQUERULE = 'P'
```

How do I get the list of affected programs due to a table change?

```
SELECT GRANTEE "PROGRAM", TTNAME "TABLE NAME", SELECTAUTH "SELECT",
      INSERTAUTH "INSERT", UPDATEAUTH "UPDATE", DELETEAUTH "DELETE"
  FROM SYSIBM.SYSTABAUTH
WHERE GRANTEETYPE= 'P'
   AND TCREATOR= <TABLE CREATOR NAME>
   AND TTNAME  = <TABLE NAME>
ORDER BY PROGRAM
```

To get the affected Plan names if an index is dropped

```
SELECT BNAME, BCREATOR, BTYPE, DNAME
  FROM SYSIBM.SYSPLANDEP
 WHERE BTYPE = 'I'
   AND BNAME = <Index name to be Dropped>
```

SYSPLANDEP table records the dependencies of plans on tables, views, aliases, synonyms, table spaces, indexes, functions, and stored procedures.

BNAME - The name of an object the plan depends on.

BTYPE - Type of object identified by BNAME:
A Alias, F User-defined function or cast function, I index, S Synonym
T table, P Partitioned table space, R-Table space, V- Views, O-Stored Procedure

DNAME - Name of the plan.

For package dependency, query SYIBM.SYSPACKDEP.

To get the view definition

```
SELECT NAME, CREATOR, SEQNO, TEXT
  FROM SYSIBM.SYSVIEWS
 WHERE NAME = <VIEW NAME>
```

SEQNO - IF THE VIEW DEFINITION IS MORE THAN ONE LINE

TEXT - VIEW DEFINITION

To get the Table/VIEW/Alias details

```
SELECT NAME, TYPE, CREATOR, COLCOUNT, TBCREATOR, TBNAME,
       KEYCOLUMNS, CREATEDTS, ALTEREDTS, CREATEDBY
   FROM SYSIBM.SYSTABLES
  WHERE NAME = <TABLE or View or Alias Name>
    AND CREATOR = < TABLE or View or Alias Creator>
```

TYPE = T (Table) or A (Alias) V (View) G (Temporary Table)

COLCOUNT - NO OF COLUMNS - only for Tables AND Views (0 for Alias)

RECLENGTH - Record length - only for Tables (0 for Alias and Views)

TBCREATOR (FOR ALIAS) - ACTUAL TABLE CREATOR
TBNAME (FOR ALIAS) - ACTUAL TABLE NAME
KEYCOLUMNS - No of Primary Keys

How do I get all the objects used by a plan?
SELECT BNAME, BTYPE FROM SYSIBM.SYSPLANDEP WHERE DNAME=PLAN-NAME

How do I get all the indexes for the table?
SELECT NAME, COLCOUNT, UNIQUERULE, CLUSTERING FROM SYSIBM.SYSINDEXES
WHERE TBNAME=TABLE-NAME
UNIQUERULE – 'P' (Primary) 'U' (Alternate)
CLUSTERING – YES for clustering index.

Give an example for Self Join
Employee table has Employee No, Employee Name, Manager Name and other details of employee. I want to know all the employees and their managers.

```
SELECT A.EMPNAME, B.MANAGERNAME
  FROM EMPTABLE A, EMPTABLE B
  WHERE A.MANAGERID = B.EMPNO
```

How do I get the difference between two timestamps in number of seconds?

```
SELECT (((DAYS (TS2) - DAYS (TS1)) * 86400) +
          ((HOUR (TIME (TS2)) - HOURS (TIME TS1))) * 3600) +
          ((MINUTE (TIME (TS2)) - MINUTE (TIME (TS1))) * 60) +
          (SECOND (TIME (TS2)) - SECOND (TIME (TS1))))
 FROM  SYSIBM.SYSDUMMY1;
```

In the first part, you subtract the start time from the end time, deriving the number of days between the timestamps. You then multiply the number of days by 86400, the seconds in a day, to convert the days to the number of seconds.

```
SELECT (DAYS (TS2) - DAYS (TS1)) * 86400
```

In the second part, you derive the additional time (hours, minutes, and seconds, list in hhmmss format) between the timestamps and convert the time to the number of seconds. First you need to extract and convert the hours to seconds:

```
SELECT (HOUR (TIME (TS2)) - HOURS (TIME TS1))) * 3600
```

Next you need to extract and convert the minutes to seconds:

```
SELECT (MINUTE (TIME (TS2)) - MINUTE (TIME (TS1))) * 60
```

Then you extract the remaining seconds:

```
SELECT SECOND (TIME (TS2)) - SECOND (TIME (TS1))
```

Lastly, you add all partial results to produce the difference between the two timestamps in total number of seconds.

Selected DB2-command panel commands: (Option 7 of DB2I)

-DIS DB(DSNDB01)
Displays all the components in the Database DSNDB01 and their status.

-DIS DB(DSNDB01) SPACENUM(xxx)
Displays information about specific table space/index space(xxx) of database

-STA DB(DSNDB01)
Starts the database DSNDB01

-STA DB(DSNDB01) SPACENUM(xxx)

Start a specific tables pace/index space of a database.

SPUFI- SQL Processing Using File Input.

Option 1 of DB2I panel takes you to SPUFI panel.

```
 DSNESP01                          SPUFI                     SSID: DSN
 ===>
 Enter the input data set name:   (Can be sequential or partitioned)
  1 DATA SET NAME..... ===> 'SMSXL86.DB2.SPUFI.INPUT(MEM1)'
  2 VOLUME SERIAL..... ===>       (Enter if not cataloged)
  3 DATA SET PASSWORD. ===>       (Enter if password protected)

Enter the output data set name:   (Must be a sequential data set)
  4 DATA SET NAME..... ===> 'SMSXL86.DB2.SPUFI.OUTPUT'

Specify processing options:
  5 CHANGE DEFAULTS... ===> Y     (Y/N - Display SPUFI defaults panel?)
  6 EDIT INPUT........ ===> Y     (Y/N - Enter SQL statements?)
  7 EXECUTE........... ===> Y     (Y/N - Execute SQL statements?)
  8 AUTOCOMMIT........ ===> Y     (Y/N - Commit after successful run?)
  9 BROWSE OUTPUT..... ===> Y     (Y/N - Browse output data set?)

 For remote SQL processing:
 10 CONNECT LOCATION   ===>


 PRESS: ENTER to process      END to exit      HELP for more information
```

'SMSXL86.DB2.SPUFI.INPUT(MEM1)' – It is my dataset where I am going to code SQL statements I want to execute. If I already have my query in this dataset, then I will set EDIT INPUT option as 'N'.

'SMSXL86.DB2.SPUFI.OUTPUT' - Output of the SQL statement is received here. We no need to allocate the dataset.

If AUTOCOMMIT is 'Y', then all the changes made to DB2 using SQL statements are committed on successful execution of the query.

If you set the option CHANGE DEFAULTS to 'Y', then you will get the screen with SPUFI default values. The SPUFI allows you to set the values for:

1. Isolation level and Maximum number of lines to be returned from a SELECT
2. Output dataset characteristics.
3. Output format characteristics. (Defining length of numeric and character column, defining headers in your output file)

QMF (Query Management Facility)

Most of the installations use SPUFI for pre-testing the SQL statements.

SPUFI comes with DB2 product. Some installation use QMF for their SQL operations. QMF provides greater flexibility and lot of options in generating the result of the query in a formatted output of report style.

Cobol-DB2 Compilation JCL

```
//APIJ1DB JOB (TVD2TM,3B42),COND=(4,LT),
//            MSGCLASS=Y,CLASS=X,NOTIFY=&SYSUID
//*******************************************************************
//* INSTREAM PROCEDURE FOR COMPILATION - PRECOMPILE STEP
//*******************************************************************
//PRECOMP EXEC PGM=DSNHPC,PARM='HOST(COBOL)',REGION=4096K
//DBRMLIB DD DISP=OLD,DSN=APIJ1.TSTINDIA.DBRMLIB(PSELECT2)
//STEPLIB DD DISP=SHR,DSN=D2TB.DSNEXIT
//         DD DISP=SHR,DSN=D2TB.DSNLOAD
//SYSIN  DD DISP=SHR,DSN=APIJ1.TSTINDIA.JCL(PSELECT2)
//SYSCIN DD DISP=(NEW,CATLG,CATLG),
//          DSN=APIJ1.PRECOMP.LIST,UNIT=SYSDA,
//          SPACE=(800,(500,500))
//SYSLIB DD DISP=SHR,DSN=APIJ1.TSTINDIA.COBOL
//       DD DISP=SHR,DSN=APIJ1.TSTINDIA.DCLGEN
//SYSPRINT DD SYSOUT=*
//SYSTERM DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSUT1 DD SPACE=(800,(500,500),,,ROUND),UNIT=SYSDA
//SYSUT2 DD SPACE=(800,(500,500),,,ROUND),UNIT=SYSDA
//*
//*******************************************************************
//* INSTREAM PROCEDURE FOR COMPILATION - COMPILE STEP
//*******************************************************************
//COBCOMP EXEC PGM=IGYCRCTL,COND=(4,LT,PRECOMP)
//SYSIN DD DSN=&&DSNHOUT,DISP=(OLD,DELETE)
//SYSLIN DD DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,
//          SPACE=(800,(&WSPC,&WSPC))
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSUT1 DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=SYSDA
//SYSUT2 DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=SYSDA
//SYSUT3 DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=SYSDA
//SYSUT4 DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=SYSDA
//SYSUT5 DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=SYSDA
//SYSUT6 DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=SYSDA
//SYSUT7 DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=SYSDA
//*******************************************************************
//* INSTREAM PROCEDURE FOR COMPILATION - LINKEDIT STEP
//*******************************************************************
//LKED EXEC PGM=IEWL,PARM='XREF',
//           COND=((4,LT,COBCOMP),(4,LT,PRECOMP))
//SYSLIB DD DSN=SYS2.COB2LIB,DISP=SHR
//       DD DISP=SHR,DSN=D2TB.DSNLOAD
//SYSLIN DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//SYSLMOD DD DISP=OLD,DSN=&USER..TSTINDIA.LOADS(&MEM)
```

```
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSUT1 DD SPACE=(1024,(50,50)),UNIT=SYSDA
//          PEND
//*
//*******************************************************************



//*INVOKING INSTREAM PROCEDURE FOR COMPILING COBOL-DB2 PROGRAM.
//*******************************************************************

//COMP  EXEC DSNHCOB,
//       MEM=DYNSQL02,
//       USER=APIJ1,
//       PARM.PRECOMP='HOST(COBOL),QUOTE,SOURCE,XREF',
//       PARM.COBCOMP='QUOTE,SOURCE,NODYNAM',
//       PARM.LKED='LIST,XREF'
//PRECOMP.SYSIN   DD DSN=&USER..TSTINDIA.COBOL(&MEM),DISP=SHR
//PRECOMP.DBRMLIB DD DSN=&USER..TSTINDIA.DBRMLIB(&MEM),DISP=SHR
//PRECOMP.SYSLIB  DD DSN=&USER..TSTINDIA.DCLGEN,DISP=SHR
//LKED.SYSLMOD    DD DSN=&USER..TSTINDIA.LOADS(&MEM),DISP=SHR
//LKED.SYSIN      DD *
          INCLUDE SYSLIB(DSNELI)
//*******************************************************************
//*BIND-PLAN STEP.
//*******************************************************************
//BIND  EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB DD DSN=D2TB.DSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//DBRMLIB DD DSN=APIJ1.TSTINDIA.DBRMLIB(DYNSQL02),DISP=SHR
//SYSTSIN  DD *
DSN SYSTEM(D2TB) RETRY(2)
BIND PLAN(APIJ1P1) MEMBER(DYNSQL02) -
OWNER(APIJ1) ISOLATION(CS) ACTION(REPLACE)
/*
Sample BIND Package Card and respective BIND Plan card WITH RUN command
//BIND     EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB DD DSN=D2TB.DSNLOAD,DISP=SHR
//DBRMLIB  DD DSN=DP.DBRMLIB,DISP=SHR
//SYSTSIN  DD  *
  DSN SYSTEM(DSNP)
  BIND PACKAGE(SOCCOLL)  MEMBER(SOC02P01)  OWNER(SOC)  +
       ACTION(REPLACE)      VALIDATE(BIND)        ISOLATION(CS)
  END
//*
//STEP1    EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSIN  DD  *
  DSN SYSTEM(DSNP)
  BIND PLAN(SOCBATCH)      PKLIST(SOCCOLL.*)      ACTION(REPLACE) +
       RETAIN              ACQUIRE(USE)           RELEASE(COMMIT) +
       ISOLATION(CS)       VALIDATE(BIND)         NODEFER(PREPARE) +
       CACHESIZE(0)
  END
```

```
//*
//JS01    EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//STEPLIB DD DSN=D2TB.DSNLOAD,DISP=SHR
//SYSTSIN   DD *
  DSN SYSTEM(DSNP)
  RUN PROGRAM(SOC02P01) PLAN(SOCBATCH)  LIB('DP.GOLIB')
  END
//
```

Interview Questions

1. Explain the steps involved in executing the coded DB2 program. *****
2. What is PLAN and PACKAGE? *****
3. Advantage of Package? ****
4. What is isolation level? Explain CS RR UR. ****
5. –818 –811 –904 –911 –205 –305 100 SQLCODES ****
6. What is Deadlock? ****
7. Difference between UNION and JOIN. ***
8. Difference between INNER JOIN and OUTER JOIN. ***
9. Difference between GROUP BY and ORDER BY ***
10. Difference between WHERE and HAVING ***
11. What is correlated sub-query?
    How it differs from non-correlated sub-query? ***
12. What is collection-id? **
13. Have you worked on performance improvement of SQL queries?
    If yes, how do you analyze the performance of the query? ***
14. What is EXPLAIN? **
15. What is NULL indicator? What does –1, -2 ,0 values mean? **
16. What are the two types of indexes in DB2? ***
17. Different Table spaces – Properties **
18. Difference between ALIAS and SYNONYM. ***
19. When can you say that a query will return only one row? *
20. What is maximum size of table space? *
21. Difference between Lock and Latch *
22. What is deadlock? **
23. Delete rules of referential integrity. **
24. Difference between Triggers and stored procedures? **
25. Queries on SYSIBM tables for properties of object. ***