What is Table?

Tables are **database objects that contain all the data in a database**. In tables, data is logically organized in a row-and-column format similar to a spreadsheet

PF created in DDS is equivalent to table created in DDL/SQL?

Yes

Creating Tables in DB2:-

Create table table name

({field1 Name } {Data type} {length} { **Value in case of null}**,

{field2 Name } {Data type} {length} { **Value in case of null}**,

{field3 Name } {Data type} {length} { **Value in case of null}**,

……..

Primary key (field name)

) rcdfmt recordformatname**(Note record format is optional in case you need to use RPG Opcodes we define record formats)**

Data types in SQL

a. INT
b. Decimal (total digits, decimal places)

   example decimal (10,3) 9999999.999

c. Numeric (total digits, decimal places)

d. CHAR(Length) :Fixed length string (Total column length can be from 0 to 255)

e. Varchar(length) : This is variable length string data type. Memory can be saved if it contains huge length of data. (Total column length can be from 0 to 32704)

f. Timestamp

There are other data types like CLOB/DLOB/Graphics etc.

g. DATE data type.

h. Time

Links for reference on data types:-

https://www.ibm.com/docs/en/db2-for-zos/12?topic=columns-numeric-data-types

https://www.ibm.com/docs/en/db2-for-zos/12?topic=columns-string-data-types

https://www.ibm.com/docs/en/db2-for-zos/12?topic=columns-date-time-timestamp-data-types

To add/Delete/Update records in data base files using SQL:-

a. **Insert query: - Adding data in database**

```
Insert into table name (column1 , column2) values(value1,value2)
```

If you need to insert values in entire table then no need to specify column names

```
INSERT into table name   values (value1, value2, ……)
```

Inserting data by selecting from another table.

```
Insert into table name
  Select columns from   source table where condition
```

Example: -

```
insert into deptpbk
   select * from deptpf where edept = 'FINANCE'

INSERT INTO DEPTBK
   SELECT FIELD1 , FIELD2,FIELD3,FIELD4,FIEDL5 FROM DEPTPF WHERE
EDEPT = ‘FINANCE’
```

Syntax:

```
Insert into table
```

```
Select fields from source table where condition
```

b. Update query: - Updating data in database.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

c. **Delete query: - Deleting data from database**

```
DELETE FROM table_name WHERE condition;
```

d. To select data from data base

```
Select column1, column2 from table where condition
```

```
Or
Select * from table where condition. (* will select all the columns
from table/file)
```

select * from emppf

select * from emppf where eDEPT = 'IT'

Select with multiple fields in where conditions.

SELECT * FROM emppf WHERE eDEPT = 'IT' AND ECITY = 'HYDERABAD' AND

ESAL > 2000

- Greater than

< Less than

>= Greater than –

< = equal to

IN Operator in SQL

It allows to specify multiple values in where clause.
Basically, this is combination of multiple OR statements on same field.

Example:

SELECT * FROM DEPTPF WHERE DEPTID IN (100,101)

OR equivalent of above query

SELECT * FROM DEPTPF WHERE DEPTID = 100 OR DEPTID = 101

EXISTS Operator in SQL

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

```
SELECT column name(s)
FROM table name
WHERE EXISTS
(SELECT column name FROM table name WHERE condition);
```

Exists example:

```
SELECT SupplierName
FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierI
D = Suppliers.supplierID AND Price < 20);
```

Data can be also be inserted/updated using UPDDTA AS400 command. Most of the companies even use DBU external 3rd party **tool for data manipulation**

Index and Views in DDL -

Logical file created in DDS can be equivalent to Index/views.

Index: These are created on data base tables to get values from data base table.

Equivalent to logical files.

Syntax: -

```
Create index index name on table(fields)
```

```
Unique index: -
Create unique index index name on table(fields)
```

Indexes are basically used for data base performance improvement by SQL optimizer on AS400

Views

In SQL, a view is a virtual table based on the result-set of an SQL statement.

SQL statement can contain one table or combination of tables (Joins /UNION)

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

View created by joining 2 or more tables:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name1 join Table_NAME2 on (common field =  common field
and….)
WHERE condition;
```

Note: A view always shows up-to-date data! The database engine recreates the view, every time a user queries it.

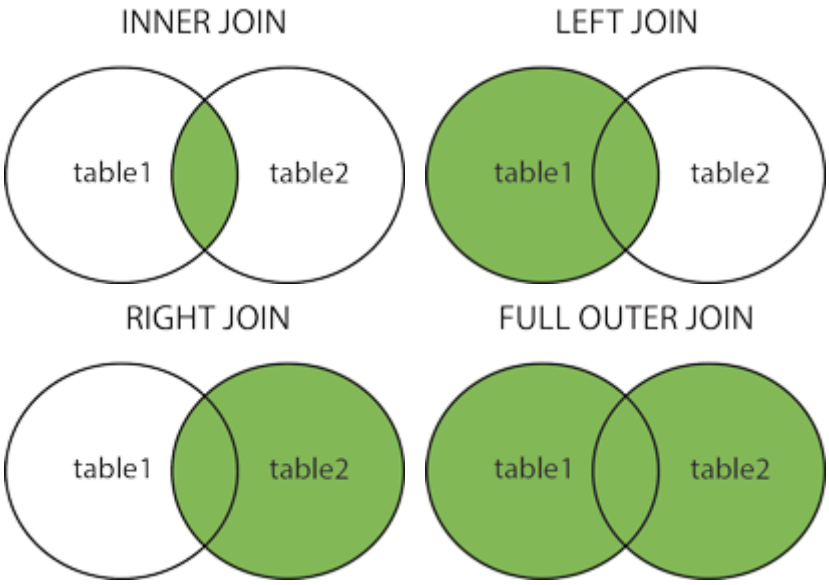
Joins in SQL: -

A `JOIN` clause is used to combine rows from two or more tables, based on a related column between them

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- `(INNER) JOIN`: Returns records that have matching values in both tables
- `LEFT (OUTER) JOIN`: Returns all records from the left table, and the matched records from the right table
- `RIGHT (OUTER) JOIN`: Returns all records from the right table, and the matched records from the left table
- `FULL (OUTER) JOIN`: Returns all records when there is a match in either left or right table



Self join : A table can be joined with itself based on related column.

Example to get Manager's name for employees based on EMPLOYEE table which has employee name, Employee id and Manager ID as columns

SELECT A.EID, A. NAME AS MANGERNAME FROM EMPMGR A

JOIN EMPMGR B ON (A.EID = BMRGID)

Creating Dependent tables in SQL using AS clause:- This is equivalent to field reference files in DDS

```
Create table table_name as
(Select field1 as fieldname in table,
 field2 as fieldname in table,
```

```
   ……..
……..
……
From table
) with no data rcdfmt rcdfmtname ;
```

Note: With data will create table by copying data from dependent table

---

Advantages of SQL: -

1.We can use arithmetic functions and create a resultant field in SQL views. Which in turn will reduce programming logic

Example :

```
CREATE VIEW STUDRESULT AS (
SELECT ROLLNO, NAME, ((ENGMARKS + MTHMARKS + SCIMARKS)/300.00)*100
 AS
PERCENTAGE FROM STUDMARKS)
```

2. We can give big meaningful names to tables/fields, same can be used in SQL queries. Normal AS400 object name and field name is limited to 10 Characters.

```
CREATE TABLE STUDENT_TABLE FOR SYSTEM NAME STABLE (
     STUDENT_ID  FOR COLUMN SID INT NOT NULL WITH DEFAULT ,
     LAST_NAME FOR COLUMN LASTNAME char(255) NOT NULL WITH DEFAULT )
RCDFMT STREC;
```

3. Relationships between tables can be maintained using foreign key. This will ensure data integrity between tables.

A foreign key is a column or group of columns in a table that uniquely identifies a row in another table. The foreign key constraints define foreign keys.

We can add this in Create table or Alter table statements.

Syntax with examples

```
FOREIGN KEY (fk1, fk2,...)
REFERENCES parent table (p1, 2,..)
ON UPDATE [ NO ACTION | RESTRICT]
```

ON UPDATE rules

Db2 triggers the ON UPDATE rule when you update a row in either parent or child table. The update rule has two options NO ACTION and RESTRICT.

When you update the row in the parent key column of the parent table, Db2 rejects the update if there is the corresponding row exists in the child table for both RESTRICT and NO ACTION option.

When you update the row in the foreign key column of the child table, Db2 rejects the update for RESTRICT option and allows the update for the NO ACTION, with the condition that the new value of the foreign key column exists in the parent table.

ON DELETE rules

Db2 triggers the ON DELETE rule when you delete a row in the parent table. Db2 determines whether or not to delete the rows in the child table based on the following options:

- NO ACTION or RESTRICT does not delete any row in both tables and issues an error.
- CASCADE deletes the row in the parent table and all related rows in the child table.
- SET NULL deletes the row in the parent table and updates values in the foreign key columns in the child table to NULL only if these columns are not nullable columns.

How to execute SQL queries in rpgle programs?

We create a SQLRPGLE source type and this allows us to add/execute sql queries in rpgle code.

If we are writing a spec form code.

In C spec use /EXEC SQL

In following lines type queries using C+

And once query is done /END-EXEC.

Example : -

```
C/EXEC SQL
C+ UPDATE TESTPF SET FIELD1 = 'SQLUPD'
C+ WHERE FIELDKEY ='duplicate'
C/END-EXEC
C                EVAL      *INLR = *ON
```

1. **Coalesce and IFNULL** functions are used to handle NON- NULL values in sql.

If we need to replace Null value in a field with equivalent non null value, we can use these functions.

Ifnull(fieldname, (non null value which will be used to replace null with)

COALESCE(fieldname, (non null value which will be used to replace null with)

Basically use these functions when left outer join is involved in View/Sql queries in SQLRPGLE.

Example :-

```
Select a.studno, a.studname, ifnull(b.percentage,0)  from
   Studtable a left outer join percentage b on (a.studno  = b.studno)
```

The SQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

ORDER BY Syntax

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

Aggregate functions and group by :

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

GROUP BY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

.

(Order by is not mandatory but its good to use as result will be in specific order)

Examples:

To get maximum price in for each order.

SELECT order no, max(price), FROM ORDERDTL

group by orderno

To get total order amount for orders at customer level:

SELECT customer, sum(amount), FROM ORDERDTL

group by customerno

Embedded/Dynamic SQL in SQLRPGLE: -

1. Make query string based on your program requirements and conditions.

Example a screen accepts

a. Roll No range
b. Class
c. Percentage range

And gets a report with that range and we need to provide option on making class as mandatory entry and other fields as optional

Prepare a dynamic sql string first.

Example

if(screenfromrollno = 0 and screentorollno = 0) and(fromper <> 0 and topercent <>0)

sqlstring = 'select * from pf where class=' + ''' + screenclass + '' + 'and percentage>=' + %char(frompercent) 'and percentage<=' + %char(topercent)

if(screenfromrollno = 0 and screentorollno = 0) and(fromper = 0 and topercent 00)

sqlstring = select * from pf where class=:screenclass

2. Prepare SQL statement from that string using prepare statement

exec sql

prepare stmt from :sqlstring

3. If you are using cursor for multiple record selection , declare cursor from that statement

exec sql

declare c1 cursor for STMT

4. If you are using dynamic update/delete/insert processing just execute that statement using execute statement

Example :

In this file name will be dynamic we will prepare sqlstring first

1. SQLSTRING = 'DELETE FROM ' + FILE

2. Prepare statement from this sql string

   Exec sql prepare S1 from :SQLSTRING

3. Execute that statement

   Exec sql EXECUTE S1

Prepare and Execute statement can be combined using

Execute immediate

The EXECUTE IMMEDIATE statement:

- Prepares an executable form of an SQL statement from a character string form of the statement
- Executes the SQL statement

```
exec sql execute immediate :SQLSTRING;
```

PARAMETER marking or Soft Coding values in sql

This is used in dynamic/embedded sql statements

Use ? in place of parameters which you need to input from parameter/variables.

Then use USING clause in OPEN cursor or PREPARE statement and use parameters/variables which you need to replace? or values in query.

Parameter marking in Cursor example:-

Substitute parameters

```
 String = 'SELECT * FROM TESTFILE WHERE ACCTNBR > ? +
                 AND ACCTSTATE = ?' ;

 exec sql PREPARE S0 FROM :String ;

exec sql DECLARE C0 CURSOR FOR S0 ;

exec sql OPEN C0 USING :wkAccount,:wkState ;

exec sql FETCH C0 INTO :InputDs ;

exec sql CLOSE C0 ;
```

Parameter marking in embedded delete query:-

Sqlstring : ‘DELETE from ’ + TABLE + ‘WHERE EMPNO =?’ ;

Exec Sql Prepare S1 from :sqlstring

Exec Sql Execute S1 using :p_empno (p_empno is parameter/variable from program)

Commitment control in SQLRPGLE:- File should be journaled for using under commitment control.

```
Columns . . . :    6 100                    Browse                    PRAKASH521/QRPGLESRC
SEU==>                                                                          COMITSQL
FMT *   *. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
        ********** Beginning of data **************************************************************************
0001.00                                                                         211228
0002.00    EXEC SQL SET OPTION COMMIT = *CHG ;                                  211014
0002.01                                                                         211228
0002.02    EXSR DATAPROCESSING ;                                                211014
0002.03    *INLR = *ON  ;                                                       211014
0003.00                                                                         211014
0003.01    BEGSR DATAPROCESSING ;                                               211228
0003.02    //HEADER                                                             211014
0004.00   EXEC  SQL                                                             211014
0005.00     INSERT INTO ORDERHEADER VALUES (1 , 100 , 'MEXICO')  ;              211014
0005.01                                                                         211014
0005.02                                                                         211014
0005.03    //DETAILS.                                                           211014
0006.00   EXEC  SQL                                                            211014
0007.00     INSERT INTO ORDERDETAIL VALUES (1 , 100 , 'MEXICO' , '12345' , 100)  ;  211014
0007.01    IF SQLCODE <  0 ;                                                    211014
0007.02                                                                         211228
0007.03   EXEC  SQL                                                            211228
0007.04       ROLlback ;                                                        211228
```

```
      LEAVESR;
   ENDIF ;

  EXEC  SQL
    INSERT INTO ORDERDETAIL VALUES (1 , 100 , 'MEXICO' , '56789' , 200)  ;
   IF SQLCODE <  0 ;
      ROLBK ;
      LEAVESR;
   ENDIF ;

  EXEC  SQL
    INSERT INTO ORDERDETAIL VALUES (1 , 100 , 'MEXICO' , '88888' , 100)  ;
   IF SQLCODE <  0 ;
      ROLBK ;
      LEAVESR;
   ENDIF ;

   COMMIT ;
```

```
 EXEC  SQL                                                                    211014
   INSERT INTO ORDERDETAIL VALUES (1 , 100 , 'MEXICO' , '88888' , 100)  ;     211014
  IF SQLCODE <  0 ;                                                           211014
     ROLBK ;                                                                  211228
     LEAVESR;                                                                 211014
  ENDIF ;                                                                     211014
                                                                              211014
  COMMIT ;                                                                    211014
                                                                              211014
 ENDSR;                                                                       211014
****************** End of data ***************************************************************************
```

Selecting distinct values in SQL :

The SQL SELECT DISTINCT Statement

The `SELECT DISTINCT` statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

Syntax:

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

Count in sql

The COUNT() function returns the number of records returned by a select query.

Example select count(*) from orderpf

This will return total number of records in ORDERPF

Selecting * means it will select every field and then count , more efficient way will be selecting particular field.

Example :

Select count(orderno) from orderpf

Or selecting any constant

Select count(1) from orderpf

UDF concepts

A *user-defined function* (UDF) is a function that is defined to the DB2® database system through the CREATE FUNCTION statement and that can be referenced in SQL statements. A UDF can be an external function or an SQL function.

We have several built in functions like count , substring etc. If we need to create our own function then it can be done using create function in AS400.

It can be external means logic for function will be in AS400 program or it can be SQL which means entire function logic will be insider function body.

In this example function accepts one parameter which is INCYMD and then returns output as character(10) , this is external function as it invokes procedure CYMDTOUSA of service program UDFSRVPGM

```
01  CREATE OR REPLACE FUNCTION MODLIB.CYMDTOUSA
                                (INCYMD DECIMAL(7,0))
02    RETURNS CHAR(10)
03    LANGUAGE RPGLE
04    DETERMINISTIC
05    NO SQL
06    EXTERNAL NAME 'MODLIB/UDFSRVPGM(CYMDTOUSA)'
07    PARAMETER STYLE GENERAL
08    PROGRAM TYPE SUB ;
```

Line 1: My function is called CYMDTOUSA, it is in MODLIB, and I need to define the parameter that is passed, inCymd.

Line 2: This function returned a 10 character value.

Line 3: This function is written in RPGLE.

Line 4: DETERMINISTIC means that if it called with the same incoming parameter more than once it somehow "remembers" the value to return.

Line 5: The code within the function does not use any SQL.

Line 6: This is where I define where the code for the function is to be found. This is given in the format: library/service program(procedure).

Line 7: This is the simplest way to pass parameters in and out of the function. Only the defined incoming parameter, line 1, and the outgoing parameter, line 2, are returned to the SQL statement.

Line 8: As the function uses a subprocedure I need to give the program type of SUB.

I run the SQL statements in the source member by using the Run SQL Statement command, `RUNSQLSTM`. This create all of my UDFs, and I am ready to use them within SQL statements, like Select, or to pass the returned value to RPG, using the Set.

Stored procedures in AS400:

These are basically used to invoke as400 process from external environments like JAVA/DOTNET etc. Parameter types are IN , OUT, INOUT

IN STANDS for input parameter which means input values will be passed in this parameter to AS400 program

OUT stands for output parameter which means output generated from AS400 side can be passed back in these kind of parameters and received at the calling side.

INOUT stands for input output parameters which means AS400 program will received input in this parameter and then whatever output is calculated/generated can be sent back in same parameter.

A *procedure* (often called a stored procedure) is a program that can be called to perform operations. A procedure can include both host language statements and SQL statements. Procedures in SQL provide the same benefits as procedures in a host language.

DB2® stored procedure support provides a way for an SQL application to define and then call a procedure through SQL statements. Stored procedures can be used in both distributed and nondistributed DB2 applications. One of the advantages of using stored procedures is that for distributed applications, the processing of one CALL statement on the application requester, or client, can perform any amount of work on the application server.

You may define a procedure as either an SQL procedure or an external procedure. An external procedure can be any supported high level language program (except System/36 programs and procedures) or a REXX procedure. The procedure does not need to contain SQL statements, but it may contain SQL statements. An SQL procedure is defined entirely in SQL, and can contain SQL statements that include SQL control statements.

Coding stored procedures requires that the user understand the following:

- Stored procedure definition through the CREATE PROCEDURE statement
- Stored procedure invocation through the CALL statement
- Parameter passing conventions
- Methods for returning a completion status to the program invoking the procedure.

You may define stored procedures by using the CREATE PROCEDURE statement. The CREATE PROCEDURE statement adds procedure and parameter definitions to the catalog tables SYSROUTINES and SYSPARMS. These definitions are then accessible by any SQL CALL statement on the system.

To create an external procedure or an SQL procedure, you can use the SQL CREATE PROCEDURE statement.

The following sections describe the SQL statements used to define and call the stored procedure, information about passing parameters to the stored procedure, and examples of stored procedure usage.

For more information about stored procedures, see Stored Procedures, Triggers, and User-Defined Functions on DB2 Universal Database™ for iSeries

- **Defining an external procedure**
  The CREATE PROCEDURE statement for an external procedure names the procedure, defines the parameters and their attributes, and provides other information about the procedure that the system uses when it calls the procedure.
- **Defining an SQL procedure**
  The CREATE PROCEDURE statement for an SQL procedure names the procedure, defines the parameters and their attributes, provides other information about the procedure that is used when the procedure is called, and defines the procedure body.
- **Defining a procedure with default parameters**
  External and SQL procedures can be created with optional parameters. Optional procedure parameters are defined to have a default value.
- **Calling a stored procedure**
  The SQL CALL statement calls a stored procedure.
- **Returning result sets from stored procedures**
  In addition to returning output parameters, a stored procedure can return a result set (that is, a result table associated with a cursor opened in the stored procedure) to the application that issues the CALL statement. The application can then issue fetch requests to read the rows of the result set cursor.
- **Writing a program or SQL procedure to receive the result sets from a stored procedure**
  You can write a program to receive results sets from a stored procedure for either a fixed number of result sets, for which you know the contents, or a variable number of result sets, for which you do not know the contents.
- **Parameter passing conventions for stored procedures and user-defined functions**
  The CALL statement and a function call can pass arguments to programs written in all supported host languages and REXX procedures.
- **Indicator variables and stored procedures**
  Host variables with indicator variables can be used with the CALL statement to pass additional information to and from a procedure.
- **Returning a completion status to the calling program**
  SQL and external procedures return status information to the calling program in different ways.
- **Passing parameters from DB2 to external proceduress**
  DB2 provides storage for all parameters that are passed to a procedure. Therefore, parameters are passed to an external procedure by address.