

Like

• The underscore sign (\_) represents any single character.

• The percent sign (%) represents a string of zero or more characters.

• Any other character represents itself.

Escape character:

For example, if the variable is defined as CHAR(10), and the value WYSE% is assigned to that variable,

the variable is padded with blanks on assignment. The pattern used is

'WYSE% '

This pattern requests the database manager to search for all values that start with WYSE and end with

five blank spaces. If you intended to search for only the values that start with 'WYSE' you should

assign the value 'WYSE%%%%%%' to the variable.

Assigning escape characters in query:

**SELECT \***

**FROM** TABLEY

**WHERE** C1 **LIKE** 'AAAA+%BBB%' **ESCAPE** '+'

'+' is the escape character and indicates that the search is for a string that starts with 'AAAA%BBB'. The

'+%' is interpreted as a single occurrence of '%' in the pattern.

One more example, can be used for multiple selections at once:

The RESUME column in sample table EMP\_RESUME is defined as a CLOB. If the variable LASTNAME has a

value of 'JONES', the following statement selects the RESUME column when the string JONES appears

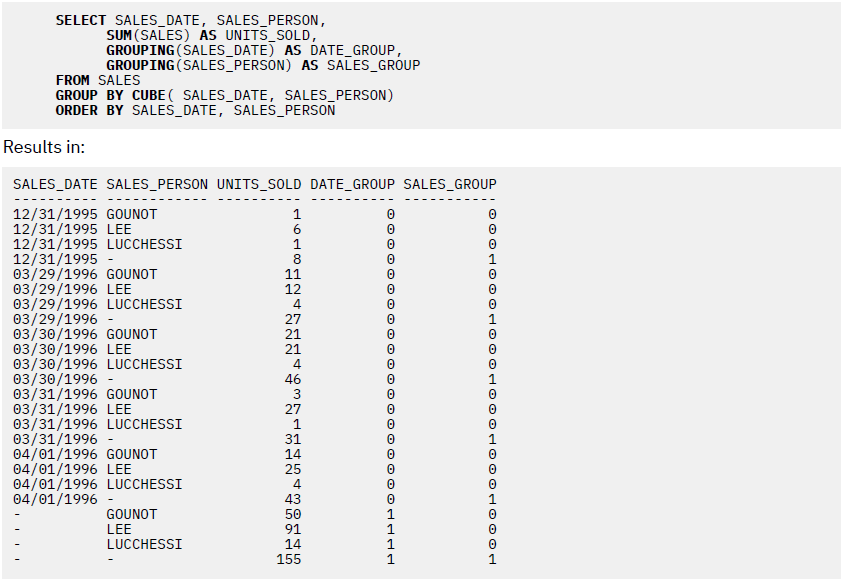
anywhere in the column.

**SELECT** RESUME

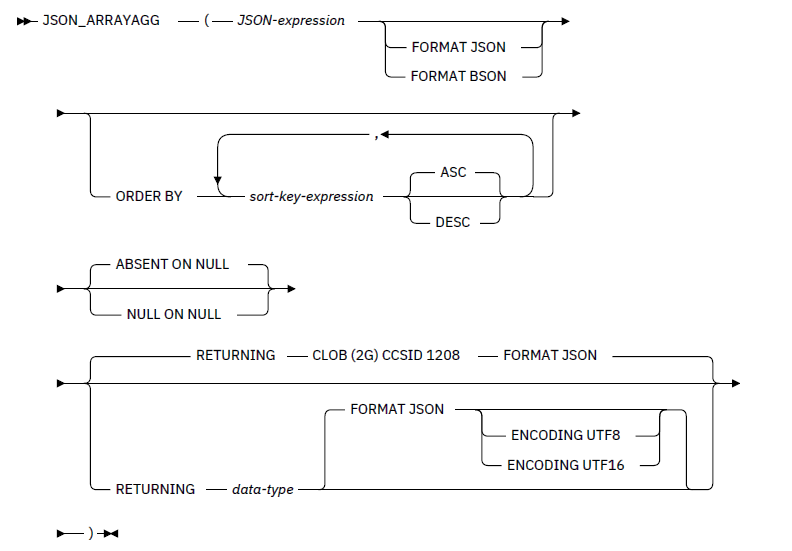
**FROM** EMP\_RESUME

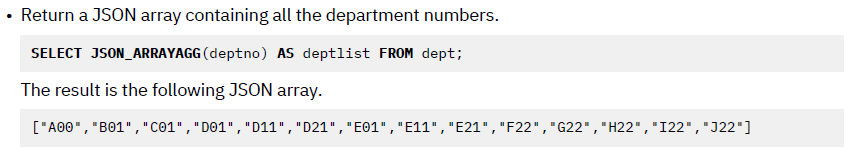
**WHERE** RESUME **LIKE** '%'||LASTNAME||'%'

Grouping Expressions

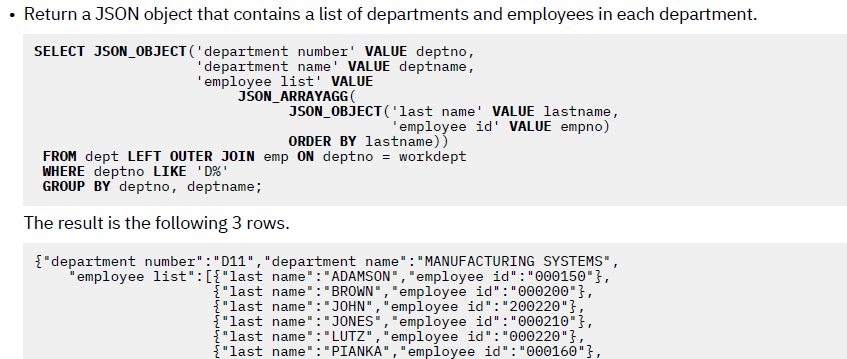
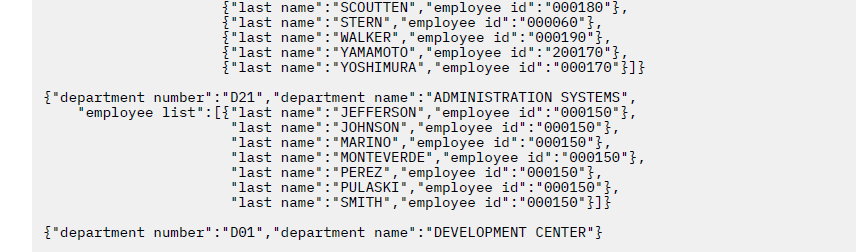


JSON\_ARRAYAGG:

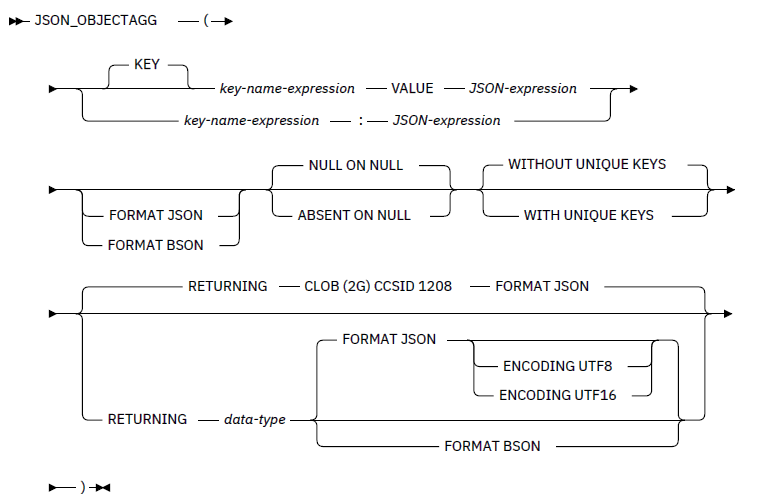


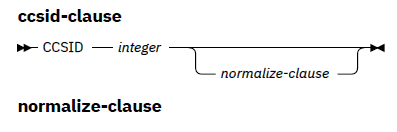


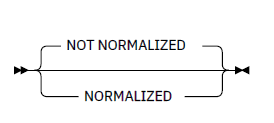
JSON\_OBJECT:

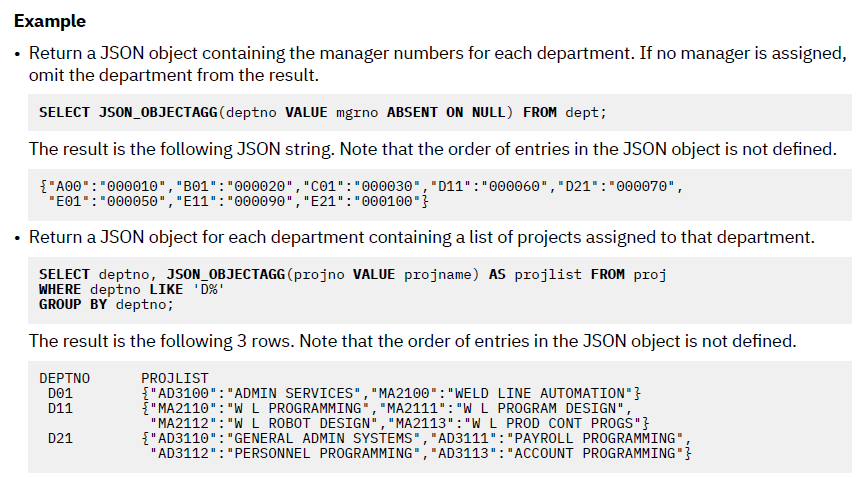
 

JSON\_OBJECTAGG:





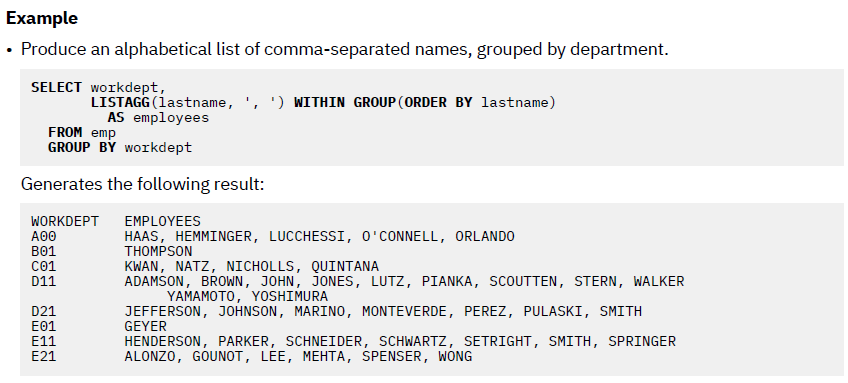




**LISTAGG**

The LISTAGG function aggregates a set of string elements into one string by concatenating the strings.

Separator is provided optionally.



XML format of outputs:

**SELECT XMLSERIALIZE**(**XMLDOCUMENT** (

**XMLELEMENT**(**NAME** "Department",

**XMLATTRIBUTES**(E.WORKDEPT **AS** "name"),

**XMLAGG**(**XMLELEMENT** ( **NAME** "emp", E.LASTNAME)

**ORDER BY** E.LASTNAME)

))

**AS CLOB**(200)) **AS** "dept\_list"

**FROM** EMPLOYEE E

**WHERE** E.WORKDEPT **IN** ('C01', 'E21')

**GROUP BY** WORKDEPT

Note: CLOB returns a character string representation

The result of the query would look similar to the following result:

dept\_list

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

<Department name="C01">

<emp>KWAN</emp>

<emp>NICHOLLS</emp>

<emp>QUINTANA</emp>

</Department>

<Department name="E21">

<emp>GOUNOT</emp>

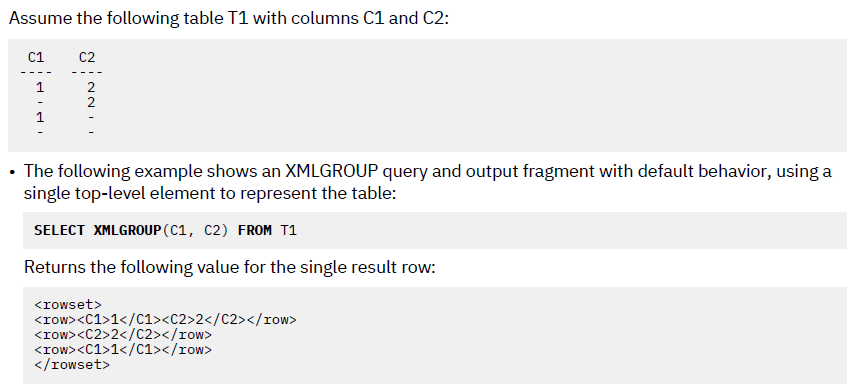
<emp>LEE</emp>

<emp>MEHTA</emp>

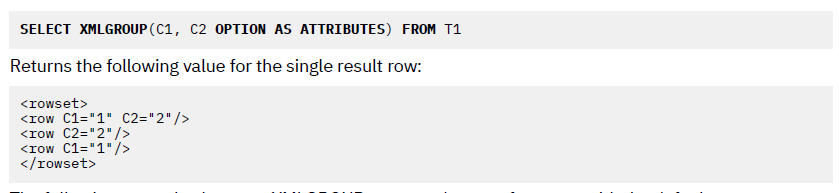
<emp>SPENSER</emp>

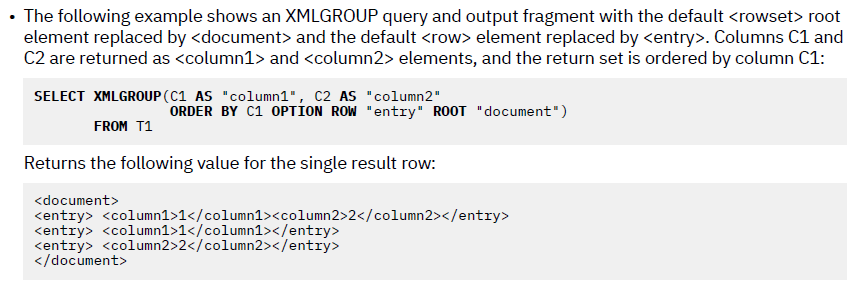
</Department>

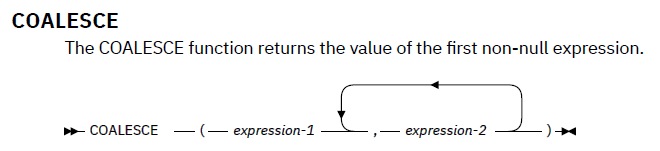
XMLGROUP



To set the column values as attributes, instead of value:

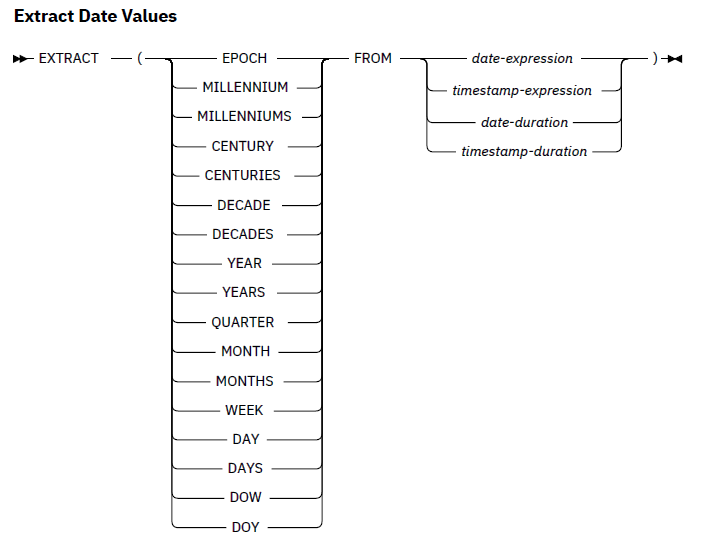


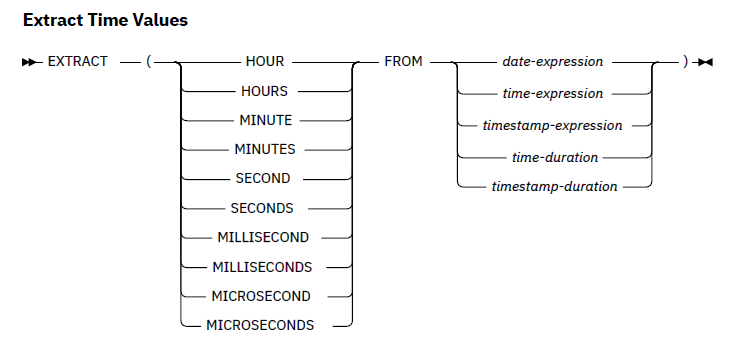




No limit in number of expressions

Whereas IFNULL can just have two expressions, first is the columns required, and second is the value to be returned if first column is null.





Cursors:

Consider the following cursor logic:

DECLARE csr1 SENSITIVE STATIC SCROLL CURSOR

FOR SELECT FIRSTNAME, LASTNME

FROM DSN8710.EMP

ORDER BY LASTNME

FETCH FIRST 1000 ROWS ONLY;

OPEN csr1;

FETCH ABSOLUTE 200 csr1 INTO :FN, :LN;

I used the FETCH FIRST 1000 ROWS ONLY clause to ensure that no more than 1,000 rows were returned. This clause is, of course, optional (and if not specified, DB2 will not limit the result set returned by the cursor). Then I open the cursor and FETCH row 200. This positions the cursor just after the 200 result row that was just fetched. After that, all you would need would be to create a loop that just issues FETCH NEXT 300 times and that would retrieve only rows 200 through 500.

Basically, scrollable cursors reduce the amount of time and effort required to move backward and forward through the results of SQL queries. But as helpful as scrollable cursors are, do not make every cursor a scrollable cursor. Scrollable cursors require substantially more overhead than a traditional, non-scrollable cursor. Analyze the requirements of your applications and deploy scrollable cursors only where it makes sense to do so.

Update file fields using declare cursor:

Restrictions: The select statement should not have a **group by** clause

**For Update of** is not mandatory unless **For read only** or **order by** clause is not mentioned, provided the cursor is not scrollable.

If cursor is scrollable, **For Update of** is mandatory.

For end of data handling in sqlrpgle, WHENEVER statement is used.

EXEC SQL WHENEVER NOT FOUND GO TO ..<tag>..;

EXEC SQL WHENEVER SQLERROR GO TO ..<tag>..;

EXEC SQL WHENEVER SQLWARNING GO TO ..<tag>..;

EXEC SQL WHENEVER SQLWARNING CONTINUE;

(not used in full free format since, tag is not compatible)