GUIDE SHARE EUROPE UK REGION

GSE UK Conference 2019
Dock into the Dark Side

Db2 12 SQL Enhancements

Chris Crone
IBM

November 2019
Session A

Place your custom session QR code here. Please remove the border and text beforehand.

Agenda

- Overview of the Family
- Review of Db2 11 Capabilities Updated in Db2 12
- Db2 12
- Summary

Overview of the Family

GUIDE SHARE EUROPE UK REGION

Db2 SQL

Db2 12 for z/OS and Db2 11 Linux, Unix & Windows (2016)

(not exhaustive, some features may be missing)

z

Multi-row INSERT, FETCH & multi-row cursor UPDATE, dynamic scrollable cursors, GET DIAGNOSTICS, join across encoding schemes, FETCH CONTINUE, SELECT FROM MERGE, routine versions, time zone support, transparent archive query, accelerated tables, **trigger versions, temporal enhancements, more built-in functions**

common

Inner and outer joins, table expressions, subqueries, GROUP BY, complex correlation, global temporary tables, CASE, 100+ built-in functions including SQL/XML, limited fetch, insensitive scrollable cursors, UNION everywhere, MIN/MAX single index, self-referencing updates with subqueries, sort avoidance for ORDER BY, row expressions, CALL from trigger, statement isolation, range partitioning, 2M statement length, GROUP BY expression, sequences, scalar fullselect, materialized query tables, common table expressions, recursive SQL, CURRENT PACKAGE PATH, VOLATILE tables, star join, sparse index, qualified column names, multiple DISTINCT clauses, ON COMMIT DROP, transparent ROWID column, FOR READ ONLY, KEEP UPDATE LOCKS, SET CURRENT SCHEMA, client special registers, long SQL object names, SELECT FROM INSERT, UPDATE or DELETE, INSTEAD OF trigger, SQL PL in routines, file reference variables, XML, FETCH FIRST & ORDER BY in subselect/fullselect, EXCEPT, INTERSECT, MERGE, BIGINT, caseless comparisons, not logged tables, text search functions, spatial, data compression, DECFLOAT, optimistic locking, ROLE, TRUNCATE, index & XML compression, created temps, inline LOB, administrative privileges, implicit cast, datetime enhancements, currently committed, moving sum & average, index include columns, row and column access controls, time travel query, trusted contexts, global variables, GROUPING SETS, ROLLUP, CUBE, DROP COLUMN, user-defined array types, Xquery, **IS NOT DISTINCT FROM, TRANSFER ownership, OFFSET clause, SQL PL constants, BINARY/VARBINARY, obfuscation**

luw

Updateable UNION in views, more built-in functions, SET CURRENT ISOLATION, MDC, XML enhancements, additional user-defined types (row and cursor), MODULEs, BOOLEAN, column organized tables (BLU Acceleration), parameter defaults, **user-defined aggregate functions, INT2, INT4, INT8, FLOAT4, FLOAT8, BPCHAR, OVERLAPS predicate**

GUIDE SHARE EUROPE UK REGION

GSE UK Conference 2019
Dock into the Dark Side

Db2 11 - Review

GUIDE SHARE EUROPE
UK REGION

GSE UK Conference 2019
Dock into the Dark Side

SQL PL Array Data Type

Array Data Type Overview

Db2 for z/OS introduces support for Arrays inside SQL Routines (UDFs or Procedures).

- Prior to Db2 11, users might have utilized the following mechanisms to pass data to/from Routines:
  - DECLARED GLOBAL TEMPORARY TABLE,
  - Concatenated Strings,
  - Long lists of parameters, or …
- The Array Data Type consists of an ordered set of elements having the same data type (length, precision, scale, CCSID as appropriate)
  - Two Types of Arrays are Supported:
    - Ordinary Arrays
      - Have a maximum cardinality (2 Billion)
      - The data type of the index value is INTEGER
    - Associative Arrays
      - Do **not** have a defined upper bound on the number of elements
      - The data type of the index values can be an INTEGER or character string (not a LOB)

Creating an Array Type

- CREATE Ordinary and Associative arrays via CREATE TYPE statement

Syntax

```
>>-CREATE TYPE--array-type-name--AS---built-in-type-----> 153


              .-2147483647-------.
>--ARRAY--[--|------------------|--]-------------------><
              |-integer-constant-|
              '-data-type2-------'
```

All types except XML

If data-type2 is specified, then array is an Associative Array

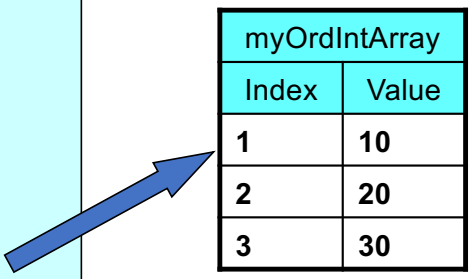
```
-- ordinary array with integer value & integer index
CREATE TYPE OrdIntArray AS INTEGER ARRAY[100];
-- associative array with integer value & integer index
CREATE TYPE AssocIntArray AS INTEGER ARRAY[INT];
-- associative array with integer value & varchar index
CREATE TYPE AssocIntArrayVarIdx AS INTEGER ARRAY[VARCHAR(13)];
```

Assigning values to Arrays

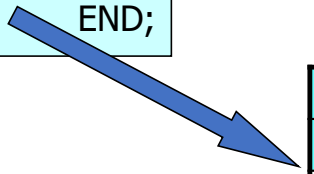
Construct arrays with array-constructor and assign values with SET assignment-statement

```
-- create a UDF that accepts and returns an ordinary array
CREATE FUNCTION UDF1 (P1 OrdIntArray, P2 INTEGER)
RETURNS OrdIntArray
BEGIN
  -- declare two ordinary array with same array type
  DECLARE myOrdIntArray OrdIntArray;
  DECLARE my2ndOrdIntArray OrdIntArray;
  -- set an ordinary array with 3 values using an array-constructor
  SET myOrdIntArray = ARRAY[10,20,30];

  -- set a 2nd ordinary array's array element, using array-element-specification
  SET my2ndOrdIntArray[1] = myOrdIntArray[3];

  ...
END;
```

```
-- ordinary array with integer value & integer index
CREATE TYPE OrdIntArray AS INTEGER ARRAY[100];
-- associative array with integer value & varchar index
CREATE TYPE AssocIntArrayVarIdx AS INTEGER ARRAY[VARCHAR(13)];
```



myOrdIntArray	
Index	Value
1	10
2	20
3	30



my2ndOrdIntArray	
Index	Value
1	30

Build an Array from Table Data

ARRAY_AGG Aggregate Function

The ARRAY_AGG function returns an array

```
CREATE TYPE OrdVchArray AS VARCHAR(12) ARRAY[100];

DECLARE myOrdVchArray OrdVchArray;

SET myOrdVchArray =
  (SELECT ARRAY_AGG(PHONE)
    FROM EMPLOYEE
    WHERE ID IS NOT NULL
    ORDER BY ID, PHONE);
```

Construct an ordinary Array

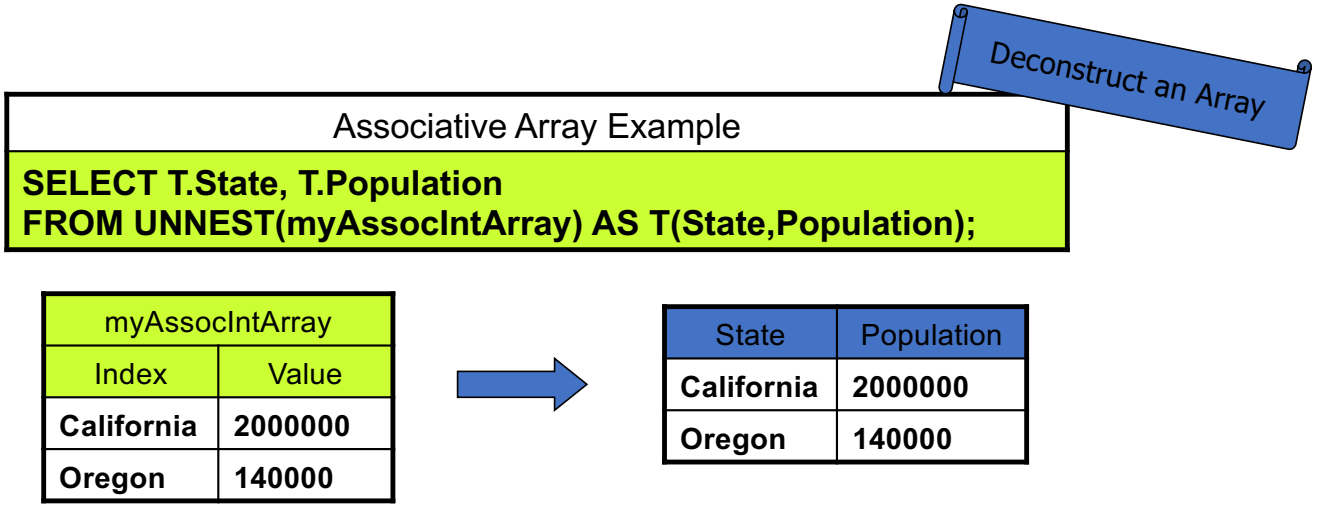
EMPLOYEE	
ID	PHONE
111222	408-555-1111
333444	408-555-3331
222333	408-555-2222
111222	408-555-1112
333444	408-555-3332

myOrdVchArray	
Index	Value
1	408-555-1111
2	408-555-1112
3	408-555-2222
4	408-555-3331
5	408-555-3332

Treating an Array Like a Table

Treat an array like a table i.e. fetch data from the array (using UNNEST collection-derived-table) just like you would from a table

- from-clause
  - collection-derived-table
    - A *collection-derived-table* can be used to convert the elements of one or more arrays into column values in separate rows of an intermediate result table



Deleting Elements From An Array

- ARRAY_DELETE Scalar Function
- The ARRAY_DELETE function deletes elements from an array.
- The result of the function has the same data type as *array-expression*.

```
-- create a procedure that accepts and returns an associative array
CREATE PROCEDURE PROC2 (IN InP1 INTEGER,
                        IN InP2 AssocIntArrayVarIdx,
                        INOUT InOutP1 AssocIntArrayVarIdx,
                        OUT OutP1 AssocIntArrayVarIdx)
BEGIN
  -- declare two associative arrays with the same array type
  DECLARE myAssocIntArray AssocIntArrayVarIdx;
  -- set an associative array with 5 values
  SET myAssocIntArray['California'] = 2000000;
  SET myAssocIntArray['Oregon'] = 140000;
  SET myAssocIntArray[New York'] = 1250000;
  SET myAssocIntArray[`New Hampshire'] = 18000;

  -- remove the state of 'Oregon' from the array
  SET myAssocIntArray = ARRAY_DELETE(myAssocIntArray,'Oregon');
  -- remove all states that begin with 'New' from the array
  SET myAssocIntArray = ARRAY_DELETE(myAssocIntArray, 'New Hampshire', 'New York');
  ...
END;
```

myAssocIntArray	
Index	Value
California	**2000000**
New Hampshire	**18000**
New York	**1250000**
Oregon	**140000**

myAssocIntArray	
Index	Value
California	**2000000**
New Hampshire	**18000**
New York	**1250000**

myAssocIntArray	
Index	Value
California	**2000000**

Finding an Array Element Index

Name	Description
ARRAY_FIRST	**returns the minimum array index value of the array**
ARRAY_LAST	**returns the maximum array index value of the array**
ARRAY_NEXT	**returns the next larger array index value for an array relative to the specified array index argument**
ARRAY_PRIOR	**returns the next smaller array index value for an array relative to the specified array index argument**

```
-- create a procedure that accepts and returns an associative array
CREATE PROCEDURE PROC2 (IN InP1 INTEGER,
                        IN InP2 AssocIntArrayVarIdx,
                        INOUT InOutP1 AssocIntArrayVarIdx,
                        OUT OutP1 AssocIntArrayVarIdx)
BEGIN
  -- declare local varchar variable
  DECLARE myVarchar, my2ndVarchar VARCHAR(20);
  -- declare two associative arrays with the same array type
  DECLARE myAssocIntArray AssocIntArrayVarIdx;
 -- set an associative array with 2 values
  SET myAssocIntArray['California'] = 2000000;
  SET myAssocIntArray['Oregon'] = 140000;

  -- sets a local variable to the index that follows 'California'
  SET myVarchar = ARRAY_NEXT(myAssocIntArray,'California');

  ...
END;
```

myVarchar = 'Oregon'

GUIDE SHARE EUROPE
UK REGION

GSE UK Conference 2019
Dock into the Dark Side

Global Variables

Global Variables Overview

- Benefits
  - Allows users to share information across SQL statements
  - Allows capability to set once, use everywhere
  - Provides additional security via Db2 GRANT and REVOKE
- Characteristics
  - Similar to special registers
  - Their **definitions** are global and shared across different connections
  - Their **contents** are only shared within the same connection
    - Each connection maintains its own instantiation
  - Their contents are NOT affected by COMMIT nor ROLLBACK
  - Use only the UNICODE encoding scheme

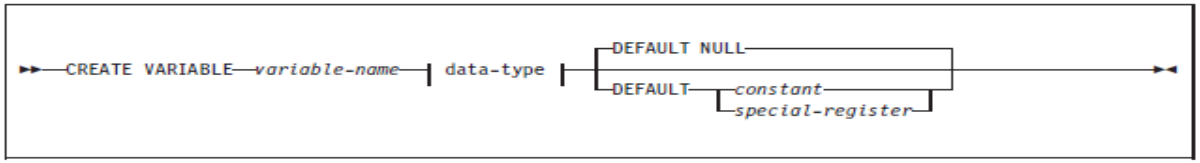
Writing and Reading Global Variables

- WRITE to via:
  - SET, SELECT INTO, VALUES INTO, OUT or inOUT parms
- READ from via:
  - Anywhere an expression can be specified
  - Exceptions
    - check constraints, MQTs, views with WITH CHECK OPTION, keys (index on expressions, XML indexes, spatial indexes), arrays
  - Values are locked in as soon as the following:
    - View definition
    - SQL scalar UDF or table UDF body
    - Trigger action
    - Row permission definition
    - Column mask definition
    - Auto-binding a package

Using and Referencing a Global Variable

GUIDE SHARE EUROPE UK REGION

Create a global variable via new CREATE VARIABLE statement



Setting and referencing a global variable

```
-- create a global variable
CREATE VARIABLE Charge_Rate DECIMAL(4,2) DEFAULT 0.00;

-- create a procedure that determines charge rate
CREATE PROCEDURE Loan_Charge_Rate (IN ID CHAR(5))
BEGIN
  ..
  SELECT SCORE INTO Cust_Score FROM CUSTOMER WHERE ACCOUNT = ID;
  IF  Cust_Score = 'Good' THEN SET Charge_Rate = 1.0;
  ELSE SET Charge_Rate = 3.0;
END;

-- calling application
myApp: PROCEDURE(Buyer, Amount);
  …
  CALL Loan_Charge_Rate(Buyer);
  UPDATE CUSTOMER SET BALANCE = BALANCE + (Amount * Charge_Rate);
END myAPP;
```

GUIDE SHARE EUROPE UK REGION

GSE UK Conference 2019
Dock into the Dark Side

Temporal Enhancements

Temporal Based Analysis

- System-maintained temporal tables
  - Db2 generated history
  - AS OF query
- User-maintained temporal tables
  - User provide time period
  - Automatic business time key enforcement.
  - Query over any current, any prior, future point/period in business time.
  - New time range update/delete statements support automatic row splitting, exploited by the merge statements.
- Bi-temporal, combination of the above two

See http://www.redbooks.ibm.com/abstracts/sg247892.html?Open Chapter 7 Application Enablement for more information

Auditing and Auditing with Temporal Example

In this example, user_id (SESSION_USER) and op_code (I/U/D) are audited. Db2 can audit a subset of special registers and session variables. The auditing is supported on regular table. When combined with temporal, it provides complete history for auditing.

```
CREATE bank_account_stt
(account_no INT NOT NULL,
 balance INT,
 user_id VARCHAR(128) GENERATED ALWAYS AS (SESSION_USER),
 op_code VARCHAR(1) GENERATED ALWAYS AS (DATA CHANGE OPERATION),
 sys_start TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN,
 sys_end TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END,
 trans_id TIMESTAMP(12) GENERATED ALWAYS AS TRANSACTION START ID,
 ...
 PERIOD SYSTEM_TIME(sys_start, sys_end));

CREATE bank_account_hist
(account_no INT NOT NULL,
 balance INT,
 user_id VARCHAR(128),
 op_code VARCHAR(1),
 sys_start TIMESTAMP(12) NOT NULL,
 sys_end TIMESTAMP(12) NOT NULL,
 trans_id TIMESTAMP(12),
 ...);

ALTER TABLE bank_account_stt ADD VERSIONING USE HISTORY TABLE
bank_account_hist ON DELETE ADD EXTRA ROW;
```

Auditing Example

At time 2, Claire inserts a row into bank_account_stt table (account no ”13452’, balance $2000);

	account	balance	User	op-code	sys_start	sys_end
base table	13452	$2000	Claire	I	2	Max
History table	(empty)					

At time 6, Steve updates the row increasing $500 balance (account_no "13452", balance $2500):

	account	balance	User	op-code	sys_start	sys_end
Base table:	13452	$2500	Steve	U	6	Max
History table:	13452	$2000	Claire	I	2	6

At time 15, Rick deletes the row of account_no "13452" (account_no "13452", balance $2500, removed from current table):

	account	balance	User	op-code	sys_start	sys_end
Base table	(empty)					
History table:	13452	$2000	Claire	I	2	6
	13452	$2500	Steve	U	6	15
	13452	$2500	Rick	D	15	15

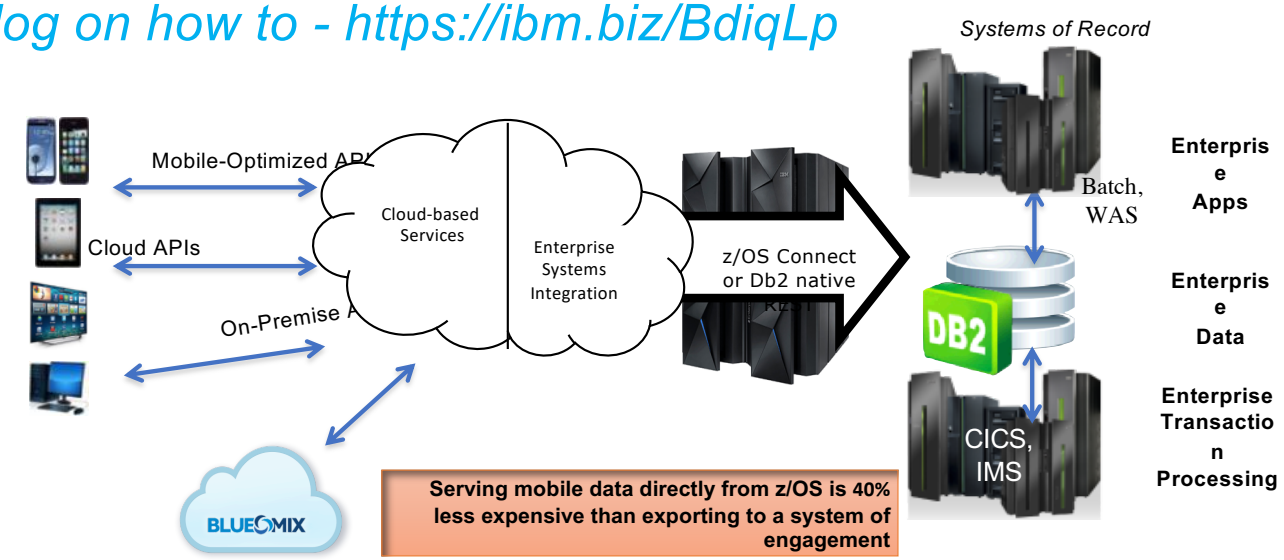
Based on history table, the row for account 13452 with balance $2000 was inserted by Claire at time 2, the balance was updated to $2500 by Steve at time 6, and was deleted by Rick at time 15 .

Db2 Data as a Service

Db2 Cloud/Mobile modernization with RESTful APIs and JSON

Blog on how to - https://ibm.biz/BdiqLp

Native Db2 REST service provider now available



- Many modern application developers work with REST services and JSON data formats
- Db2 12 (and Db2 11 APAR PI66828) ship a Native Db2 REST service
  - Easier DBA management of Db2 RESTful services, means easier adoption
  - z/OS Connect Enterprise Edition (zCEE) integration

Native REST Client Certificate Support – V11: PI80087 ; V12: PI80088
Native REST Trusted Context Support - V11: PI80087 ; V12: PI80088
Native REST Persistent Connection Support – V11: PI86867 ; V12: PI86868
Native REST TSO BIND/FREE Service Support – V11: PI86867 ; V12: PI86868
Native REST REBIND Package Support – V11: PI90243 ; V12: PI90243
Native REST DDF Profile Monitoring – V11: PI90243 ; V12: PI90243
Native REST Support for IDAA V5 – V11: PI90243 ; V12: PI90243

GUIDE SHARE EUROPE
UK REGION

GSE UK Conference 2019
Dock into the Dark Side

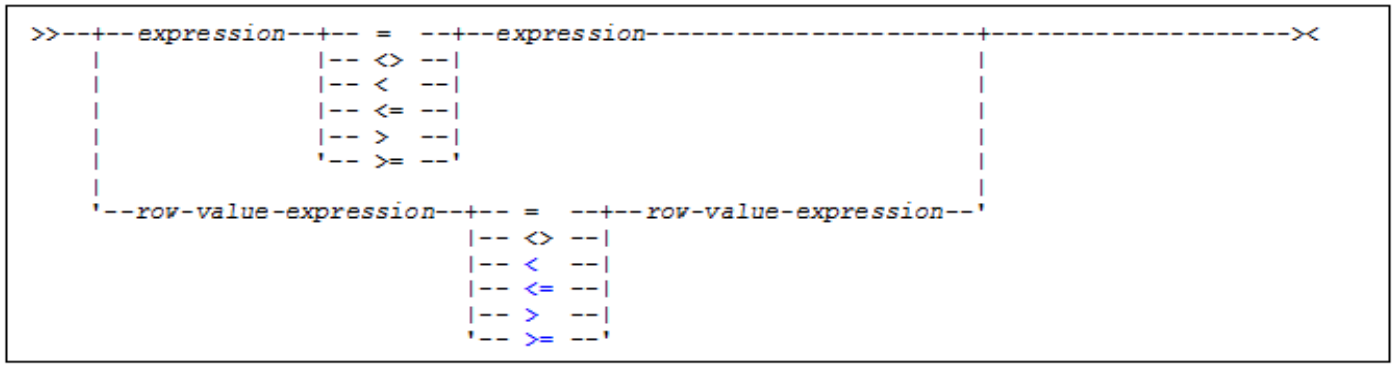
Db2 12

GUIDE SHARE EUROPE UK REGION

GSE UK Conference 2019
Dock into the Dark Side

Pagination, LIMIT, and OFFSET

Data-Dependant Pagination



- Support comparison operator <, <=, >, and >= in basic predicate with *row-value-expression*
  - Disallow non-deterministic operands
- Available when APPLCOMPAT is ‘V12R1’
- V11 allows comparison operator = and <> only

Use Cases and Considerations – Data-Dependent Pagination

- Instead of coding
  - WHERE (LASTNAME = ‘JONES’ AND FIRSTNAME > ‘WENDY’)
    OR (LASTNAME > JONES)
- You can now code
  - WHERE (LASTNAME, FIRSTNAME) > (‘JONES’, ‘WENDY’)
  - Db2 will convert to original ‘OR’ syntax
- Simplifies SQL coding – especially when number of search columns increase
- Performance expectation (original or new (C1, C2) <|<=|>=|> (:HV1, :HV2))
  - Because pagination is Data-Dependent, it may outperform the OFFSET clause
    - Must update predicate starting point to reflect the last row from the prior page
    - Data dependent pagination performance improvement from 10% from small offsets to 80-90% or more for large offsets

Result Set Positioning

Applications require to access part of Db2 result set based on a certain position

- **Numeric-based pagination** if applications need to access part of Db2 result set based on an absolute position
  - Db2 9, 10 or 11
    - SQL queries using a scrollable cursor, rowset cursor, an OLAP function, or an SQL PL routine are often used to implement numeric-dependent pagination
  - Db2 12 - **OFFSET m ROWS clause**
    - Allows Db2 to skip unwanted rows of the result set at 1st FETCH time and return the desired part of the result set, which can be an efficient way to filter unwanted rows from Db2 result set.
    - Other database systems, including Db2 LUW, support the syntax

Result Set Truncation (LIMIT)

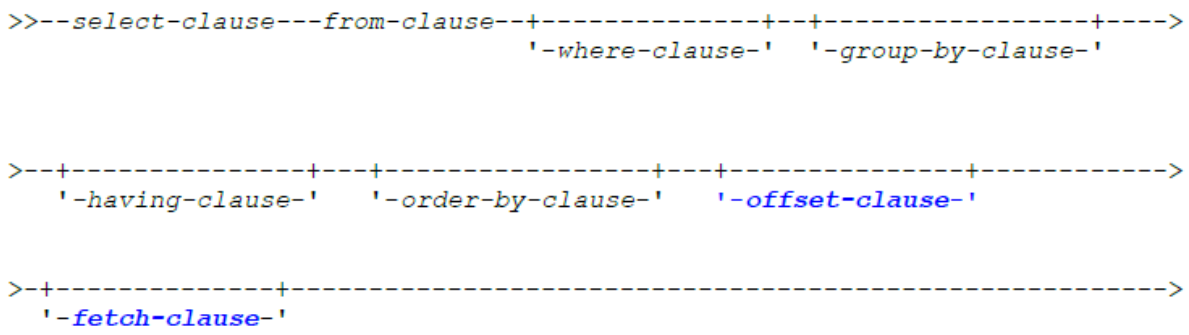
Applications require the use a variable value in the FETCH FIRST clause

- Db2 11
  - Allow INTEGER constant in FETCH FIRST clause only

    Example: **FETCH FIRST 10 ROWS ONLY**
- Db2 12
  - Allow variable in FETCH FIRST clause
    Example: **FETCH FIRST :HV1 ROWS ONLY**
  - Allow BIGINT value
    - ➔ Allow castable to BIGINT
  - Other database systems, including Db2 LUW, support the function
  - Application developers are more efficient to develop good performing mobile applications

All Function is available with APPLCOMPAT(V12R1)

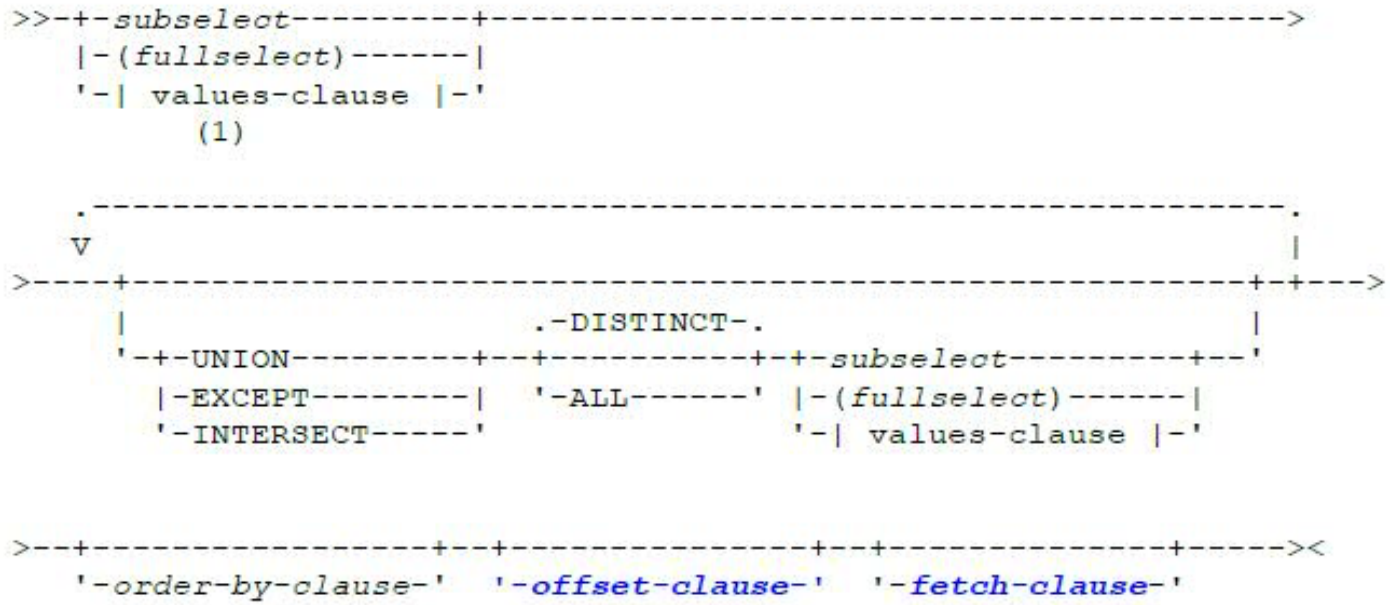
Numeric-based pagination Syntax - *subselect*



OFFSET *row-count ROWS*

- Specify the number of rows to skip from the beginning of a result set
- Allow variables and constants
  - Allow a zero or positive value; 0 means no row is skipped
- Allow castable to bigint

Numeric-based Pagination Syntax – *fullselect*



*** Also available on SELECT INTO statements

Use Cases and Considerations – Numeric-based Pagination

As you are paging forward from starting point – Prior to Db2 12

- 1st request FETCH FIRST 20 ROWS ONLY (to fill 1st screen)
- 2nd request FETCH FIRST 40 ROWS ONLY (to fill 2nd – application discards 1st 20)

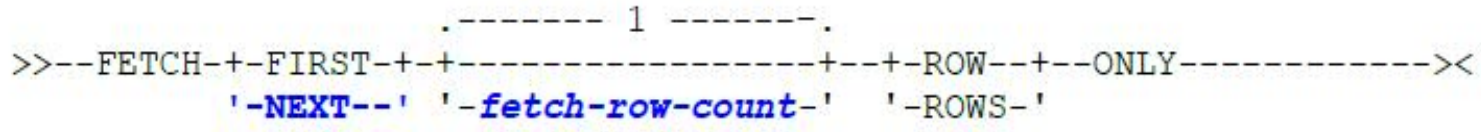
With Db2 12 - You can now code

- OFFSET 0 ROWS FETCH FIRST 20 ROWS ONLY
- OFFSET 20 ROWS FETCH FIRST 20 ROWS ONLY
- or OFFSET ? ROWS FETCH FIRST ? ROWS ONLY…

Performance expectation

- If a customer is currently using application logic to discard the first n rows:
  - 1st request FETCH FIRST 20 ROWS ONLY (to fill 1st screen)
  - 2nd request FETCH FIRST 40 ROWS ONLY (to fill 2nd – application discards 1st 20)
- Using OFFSET clause can show 10% CPU/elapsed saving for small OFFSET – with savings accumulating up to 80-90% saving for large OFFSET
  - OFFSET 0 ROWS FETCH FIRST 20 ROWS ONLY
  - OFFSET 20 ROWS FETCH FIRST 20 ROWS ONLY

FETCH Clause Enhancement

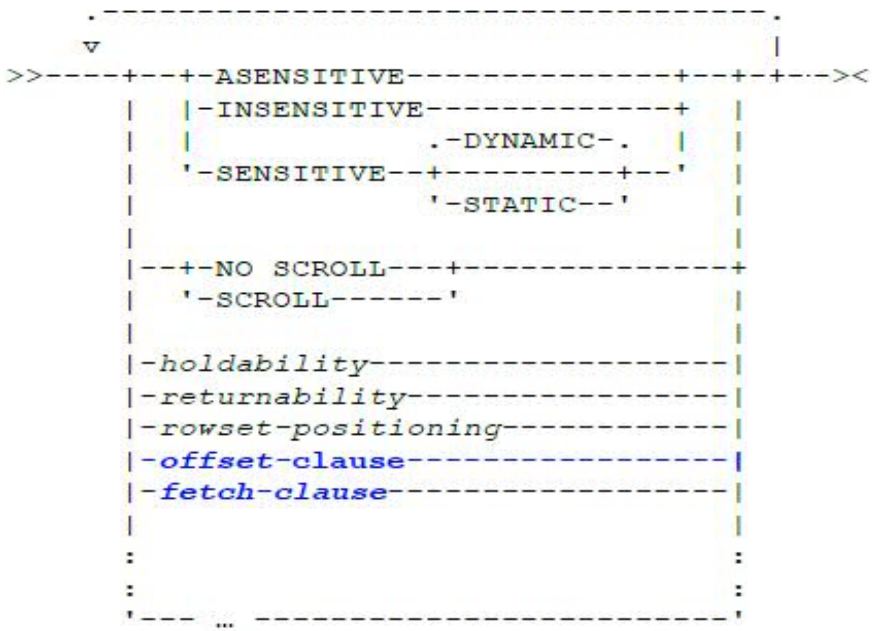


FETCH FIRST n ROWS ONLY enhancement

- Allow variables (fetch-row-count)
  - Does not allow variable in the PREPARE ATTRIBUTES string
- Allow a zero value to mean 'no row is requested'
- Allow castable to BIGINT
- New keyword NEXT interchangeable with keyword FIRST
- Disallow in the outermost fullselect for a sensitive dynamic scrollable cursor
- Allow in the outermost fullselect that contains the FOR UPDATE OF clause
- Require ORDER BY clause to ensure rows being returned are predictable
- FETCH FIRST *constant* ROWS ONLY is ignored by the CONCENTRATE STATEMENTS WITH LITERALS clause in the PREPARE *attribute-string*

Numeric-based Pagination for DYNAMIC SQL

attribute-string



Examples:

ATTRSTR = '**OFFSET 20 ROWS FETCH FIRST 4 ROWS ONLY**;

EXEC SQL PREPARE STMT1 ATTRIBUTES :ATTRSTR FROM :STMT;

LIMIT – Syntax Alternatives

Syntax alternatives for OFFSET and FETCH FIRST keywords

IBM Preferred syntax Will be reflects in the syntax diagram (ROW and ROWS are synonyms)	**Valid syntax alternatives** Will not be reflected in the syntax diagram; will only be documented as alternative syntax in a usage note
FETCH FIRST *num-rows* ROWS ONLY FETCH NEXT *num-rows* ROWS ONLY	LIMIT *num-rows*
OFFSET *offset* ROWS FETCH FIRST *num-rows* ROWS ONLY OFFSET *offset* ROWS FETCH NEXT *num-rows* ROWS ONLY	LIMIT *offset*, *num-rows*
OFFSET *offset* ROWS FETCH FIRST *num-rows* ROWS ONLY OFFSET *offset* ROWS FETCH NEXT *num-rows* ROWS ONLY	LIMIT *num-rows* OFFSET *offset*

- Syntax alternatives are available when APPLCOMPAT is ‘V12R1’

Performance Consideration

Affect Optimizer's decision on avoiding sort if OPTIMIZE FOR n ROWS is omitted

- Assume a value for FETCH FIRST *variable* ROWS ONLY
  - ➔ Subselect: 25
  - ➔ Searched DELETE: 10000

Prune the query block if FETCH FIRST 0 ROWS ONLY is specified in a subselect

- Build bind-time pruning predicate 1 = 2 if a constant
- Build run-time pruning predicate variable > 0 if a variable

Replicate FETCH FIRST n ROWS ONLY into inner query block

- Must satisfy ORDER BY and FETCH FIRST push-down rules
- Reduce the number of rows being returned from underneath subselect or fullselect
- Example:

```
SELECT C1 FROM T1
   UNION ALL
SELECT C1 FROM T2
ORDER BY 1
FETCH FIRST 10 ROWS ONLY;
```

➔

```
(SELECT C1 FROM T1
      ORDER BY 1 FETCH FIRST 10 ROWS ONLY)
   UNION ALL
(SELECT C1  FROM T2
      ORDER BY 1 FETCH FIRST 10 ROWS ONLY)
ORDER BY 1
FETCH FIRST 10 ROWS ONLY;
```

GUIDE SHARE EUROPE UK REGION

GSE UK Conference 2019
Dock into the Dark Side

ARRAY Data Type Support

Array Type and Array Variable Examples

Array Type (V11)

```
--create ordinary array type: INTEGER values, max 100 elements
CREATE TYPE OrdIntArray AS INTEGER ARRAY[100];

--create associative array type: INTEGER values, VARCHAR index
CREATE TYPE AssocIntArrayVarIdx AS INTEGER ARRAY[VARCHAR(10)];
```

SQL PL variable (V11)

```
--declare ordinary array type SQL PL variable
DECLARE VARIABLE myOrdIntAGV OrdIntArray;

--declare ordinary array type SQL PL variable
DECLARE VARIABLE myAssocIntAGV AssocIntArrayVarIdx;
```

Global variable (V12)

```
--create ordinary array type global variable
CREATE VARIABLE myOrdIntAGV OrdIntArray;

--create ordinary array type global variable
CREATE VARIABLE myAssocIntAGV AssocIntArrayVarIdx;
```

GUIDE SHARE EUROPE UK REGION

GSE UK Conference 2019
Dock into the Dark Side

Piecewise DELETE

New Syntax on DELETE

- The searched delete will be enhanced to allow the fetch-clause to be specified:

Syntax notes

1. If the *period-clause* is specified, the *fetch-clause* must not be specified (SQLSTATE 42601, SQLCODE -104).

Piece-wise Modification of Data

- Customers want to be able to mitigate the effects of locking and logging when potentially millions of rows could be affected by a simple statement like:
  "DELETE FROM T1 WHERE C1 > 7".
- Solution
  - Allow the fetch-clause to be specified on a searched delete statement

```
DELETE FROM T1 WHERE C1 > 7 FETCH FIRST 5000 ROWS ONLY;
COMMIT;
DELETE FROM T1 WHERE C1 > 7 FETCH FIRST 5000 ROWS ONLY;
COMMIT;
```

- Restrictions
  - Cannot reference a temporary table
  - Cannot reference a view that contains an INSTEAD of TRIGGER (use the base table)
  - Cannot execute this statement on the accelerator

https://www.idug.org/p/bl/et/blogid=278&blogaid=648

GUIDE SHARE EUROPE
UK REGION

GSE UK Conference 2019
Dock into the Dark Side

Enhanced Merge

Comparing Original MERGE –vs – Extended MERGE

Original MERGE

1. Source can only be host var array or list of values.
2. Single simple MATCHED and NOT MATCHED clauses.
3. Only allow one UPDATE and one INSERT operation.
4. A row in target table can be operated multiple times.
5. NOT atomic operation

   NOT ATOMIC CONTINUE… required if more than one row

Extended MERGE

1. Source can be table, view, fullselect, host var array, or list of values.
2. Allow additional predicates in MATCHED and NOT MATCHED clauses.
3. Allow DELETE as an action.
4. Allow multiple UPDATE, DELETE, and INSERT actions.
5. Allow IGNORE as an action.
6. Allow SIGNAL statement.
7. A row in target table can **only** be operated on **once** (u/d/i).
8. Atomic operation

   Not specifying “NOT ATOMIC…” signifies Extended MERGE

Example of Multi-Row MERGE (Original Merge)

```
MERGE  INTO CONTACT A
  USING  (VALUES
    ( :W300-LASTNAME, :W300-FIRSTNAME , :W300-TELEPHONE-NUMBER)
  FOR 10 ROWS) AS B
    ( LASTNAME, FIRSTNAME, TELEPHONE_NUMBER)
ON (A.TELEPHONE_NUMBER = B.TELEPHONE_NUMBER)
WHEN MATCHED THEN UPDATE    -- UPDATE when matched
   SET A.LASTNAME = B.LASTNAME, A.FIRSTNAME = B.FIRSTNAME
WHEN NOT MATCHED THEN INSERT       -- INSERT when not matched
   ( LASTNAME, FIRSTNAME, TELEPHONE_NUMBER)
      VALUES (B.LASTNAME,  B.FIRSTNAME, B.TELEPHONE_NUMBER)
NOT ATOMIC CONTINUE ON SQLEXCEPTION
```

MERGE with FULLSELECT

```
MERGE INTO ACCOUNT T
USING
 (SELECT ACCOUNT_NUMBER,  AMOUNT,  ORDER_TSTAMP FROM ORDERS)
 AS S
ON (S.ACCOUNT_NUMBER=T.ACCOUNT_NUMBER and
      S.ORDER_TSTAMP=T.ORDER_TSTAMP)
WHEN MATCHED  THEN
  UPDATE SET T.AMOUNT = S.AMOUNT
WHEN NOT MATCHED THEN
   INSERT (ACCOUNT_NUMBER, AMOUNT, ORDER_TSTAMP)
   VALUES(S.ACCOUNT_NUMBER, S.AMOUNT, S.ORDER_TSTAMP)
NOT ATOMIC CONTINUE ON SQLEXCEPTION
```

https://www.idug.org/p/bl/et/blogid=278&blogaid=664

More MATCH and NOT MATCHED Conditions Allowed

```
MERGE INTO archive ar
USING (SELECT activity, description, date, last_modified
FROM activities_groupA) ac
ON (ar.activity = ac.activity) AND ar.group = 'A'
WHEN MATCHED AND ar.last_modified < ac.last_modified THEN
 UPDATE SET
  (description, date, last_modified) = (ac.description, ac.date, DEFAULT)
WHEN MATCHED AND ar.last_modified = ac.last_modified THEN
 UPDATE SET
  (description, date,) = (ac.description, ac.date)
WHEN NOT MATCHED AND ac.date >= CURRENT DATE THEN
 INSERT
  (group, activity, description, date)
  VALUES ('A', ac.activity, ac.description, ac.date)
```

DELETE is Allowed Now

```
MERGE INTO archive ar
USING (SELECT activity, description, date, last_modified
FROM activities_groupA) ac
ON (ar.activity = ac.activity) AND ar.group = 'A'
WHEN MATCHED AND ar.last_modified < ac.last_modified THEN
 UPDATE SET
  (description, date, last_modified) = (ac.description, ac.date, DEFAULT)
WHEN MATCHED AND ar.last_modified = ac.last_modified THEN
 UPDATE SET
  (description, date,) = (ac.description, ac.date)
WHEN MATCHED AND ac.date < CURRENT DATE THEN
 DELETE
WHEN NOT MATCHED AND ac.date >= CURRENT DATE THEN
 INSERT
  (group, activity, description, date)
  VALUES ('A', ac.activity, ac.description, ac.date)
```

IGNORE and SIGNAL

```
MERGE INTO archive ar
USING (SELECT activity, description, date, last_modified
FROM activities_groupA) ac
ON (ar.activity = ac.activity) AND ar.group = 'A'
WHEN MATCHED AND ac.date IS NULL THEN SIGNAL SQLSTATE '70001'
 SET MESSAGE_TEXT =
  ac.activity CONCAT ' cannot be modified. Reason: Date is not known'
WHEN MATCHED AND ac.date < CURRENT DATE THEN
 DELETE
WHEN MATCHED AND ar.last_modified < ac.last_modified THEN
 UPDATE SET
  (description, date, last_modified) = (ac.description, ac.date, DEFAULT)
WHEN NOT MATCHED AND ac.date IS NULL THEN SIGNAL SQLSTATE '70002'
 SET MESSAGE_TEXT =
  ac.activity CONCAT ' cannot be inserted. Reason: Date is not known'
WHEN NOT MATCHED AND ac.date >= CURRENT DATE THEN
 INSERT
  (group, activity, description, date)
  VALUES ('A', ac. activity, ac.description, ac.date)
ELSE IGNORE
```

SELECT FROM MERGE

- SELECT FROM - INSERT and UPDATE Supported
  - FINAL TABLE clause MUST be specified

    SELECT FROM FINAL TABLE MERGE ….

- SELECT FROM DELETE NOT supported

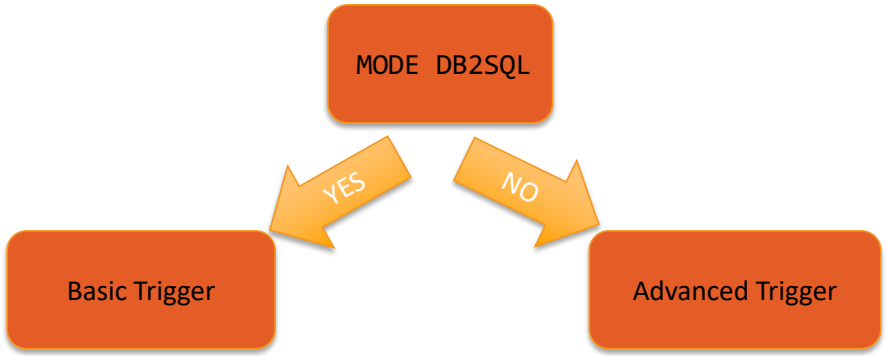
GUIDE SHARE EUROPE
UK REGION

GSE UK Conference 2019
Dock into the Dark Side

SQL PL

Basic and Advanced Triggers

- Question: How does Db2 know what type of trigger to create?
- Answer: User specified MODE DB2SQL on CREATE TRIGGER



Advanced Trigger Example

```
CREATE TRIGGER MYTRIG01
BEFORE INSERT ON MYTAB
REFERENCING NEW AS N
FOR EACH ROW
ALLOW DEBUG MODE
QUALIFIER ADMF001
WHEN(N.ending IS NULL OR n.ending > '21:00')
L1: BEGIN ATOMIC
        IF (N.ending IS NULL) THEN
            SET N.ending = N.starting + 1 HOUR;
        END IF;
        IF (N.ending > '21:00') THEN
            SIGNAL SQLSTATE '80000'
            SET MESSAGE_TEXT = 'Class ending time is
beyond 9 pm';
        END IF;
        SET GLOBAL_VAR = NEW.C1;
END L1#
```

New options can be specified

IF statement (SQL control statement)

SIGNAL statement (SQL control statement)

Enhanced SET-assignment-statement

Trigger body contains logic.

If the class end time is null, the value is set to 1 hour after the start of the class.

Otherwise, if the class ends after 9pm an error is returned.

Altering an Advanced Trigger Example

```
ALTER TRIGGER MYTRIG01
ADD VERSION V2
REFERENCING NEW AS N
FOR EACH ROW
WHEN(N.ending IS NULL OR n.ending > '22:00')
L1: BEGIN ATOMIC
  IF (N.ending IS NULL) THEN
    SET N.ending = N.starting + 1 HOUR;
  END IF;
  IF (N.ending > '22:00') THEN
    SIGNAL SQLSTATE '80000'
    SET MESSAGE_TEXT =
         'Class ending time is beyond 10 pm';
  END IF;
END L1#

ALTER TRIGGER MYTRIG01 ACTIVATE VERSION V2#
```

Add a version to an existing trigger.

For the new version, change the end time check to 10pm, instead of 9pm.

After altering the trigger, issue ALTER TRIGGER ACTIVATE with new version ID to define version V2 of the trigger as the active version of the trigger.

Replacing an Advanced Trigger Example

```
CREATE OR REPLACE TRIGGER MYTRIG01
VERSION V1
REFERENCING NEW AS N
FOR EACH ROW
WHEN(N.ending IS NULL OR N.ending > '23:00')
L1: BEGIN ATOMIC
  IF (N.ending IS NULL) THEN
    SET N.ending = N.starting + 1 HOUR;
  END IF;
  IF (N.ending > '23:00') THEN
    SIGNAL SQLSTATE '80000'
      SET MESSAGE_TEXT =
        'Class ending time is beyond 11 pm';
  END IF;
END L1#

ALTER TRIGGER MYTRIG01 ACTIVATE VERSION V1#
```

Replace trigger having VERSION V1.

For the new version, change the end time check to 11pm.

After replacing the trigger with VERSION V1, issue ALTER TRIGGER ACTIVATE with V1 to activate the updated version of the trigger (i.e. V2 was the active version before).

Modifying a trigger without affecting the activation order

- Create 3 BEFORE INSERT triggers against table T1
  - When an INSERT statement is issued against table T1, MYTRIG1 is activated then MYTRIG2 is activated then MYTRIG3 is activated

```
CREATE TRIGGER
MYTRIG1
BEFORE INSERT ON T1
VERSION V1 . . .
```



```
CREATE TRIGGER
MYTRIG2
BEFORE INSERT ON T1
VERSION V1 . . .
```



```
CREATE TRIGGER
MYTRIG3
BEFORE INSERT ON T1
VERSION V1 . . .
```

- Modify trigger MYTRIG2, keeping activation order in tact:
  - Solution 1: Issue a CREATE OR REPLACE with same version

```
CREATE OR REPLACE TRIGGER MYTRIG2 VERSION V1  . . .
```

  - Solution 2: Issue an ALTER TRIGGER REPLACE VERSION with same version

```
ALTER TRIGGER MYTRIG2 REPLACE VERSION V1  . . .
```

  - Solution 3: Issue an ALTER TRIGGER ADD VERSION with different version with a different version followed by an ALTER TRIGGER ATIVATE VERSION

```
ALTER TRIGGER MYTRIG2 ADD VERSION V2 . . .
ALTER TRIGGER MYTRIG2 ACTIVATE VERSION V2 . . .
```

Dynamic SQL in Compiled SQL Scalar Functions

- Support Dynamic SQL inside of compiled SQL scalar functions like what is supported in SQL Procedures
- Available in Db2 12 with new function activated

```
CREATE FUNCTION DYNSQLFUNC()
  RETURNS INTEGER
  VERSION V1 DETERMINISTIC NO EXTERNAL ACTION
  PARAMETER CCSID UNICODE
BEGIN
  DECLARE VARCOUNT INTEGER;
  DECLARE LV_STMT_STR VARCHAR(256);
  DECLARE S1 STATEMENT;
  DECLARE C1 CURSOR FOR S1;
  SET LV_STMT_STR = 'SELECT COUNT(*) FROM
SYSIBM.SYSTABLES';
  PREPARE S1 FROM LV_STMT_STR;
  OPEN C1;
  FETCH C1 INTO VARCOUNT;
  CLOSE C1;
  RETURN VARCOUNT;
END!
```

Constant Support in SQL Routines and Triggers

- Prior to Db2 12, inside of SQL PL, users could declare SQL variables with different data types but users were not allowed to declare any of those variables to be constant
- Db2 12 now supports the declaration of user-defined constants in SQL Routines and Triggers
  - Variables with an array type can not be declared as constant
  - Constant SQL variables are read-only
    - SQLCODE/SQLSTATE can not be declared as constant SQL variables

```
…
DECLARE VAR2 INTEGER;
DECLARE cMAXVAL INTEGER CONSTANT 2000;
SELECT 1 INTO VAR2 FROM TEST WHERE VAR1 >
cMAXVAL;
IF VAR1 > cMAXVAL THEN
  …
ELSE
  …
END IF;
```

CREATE_WRAPPED Stored Procedure

- Invokes the WRAP built-in function and executes the DDL.

```
CALL CREATE_WRAPPED('create procedure jason.P1 (inout p1
char(1))  modifies sql data language sql  begin SELECT ''A''
INTO P1 FROM SYSIBM.SYSDUMMY1; end');
```

Native SQL PL Procedure as input to CREATE_WRAPPED stored procedure

SYSIBM.SYSROUTINES

NAME	WRAPPED	TEXT
P1	Y	CREATE PROCEDURE JASON.P1 (inout p1 char(1)) WRAPPED DSN12100 ablGWmdiWmtuTmdyTmtKTmteUmdCUmdqUotKWodm1i daWmdaWmdaWntvUzcaGicaGRQ64UO_7K5gVYol9Rio jJ20Fl3zVc0lG3muMefim5vS3A4W1s6Wvf8DkcBJVz PpYG1gLxh_cUCWa

SYSIBM.SYSROUTINES

NAME	STATEMENT
P1	SELECT ‘A’ INTO :H:H FROM SYSIBM.SYSDUMMY1

GUIDE SHARE EUROPE UK REGION

GSE UK Conference 2019
Dock into the Dark Side

New Built-In Functions

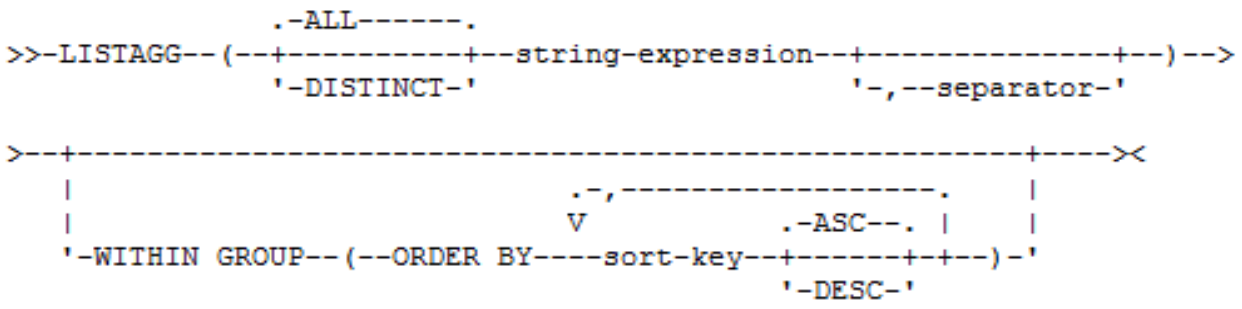
New Built-In Functions

- Aggregate Functions
  - MEDIAN
  - PERCENTILE_CONT
  - PERCENTILE_DISC

- Scalar Functions
  - GENERATE_UNIQUE_BINARY
  - VARCHAR_BIT_FORMAT
  - HASH_CRC32
  - HASH_MD5
  - HASH_SHA1
  - HASH_SHA256

LISTAGG (FL 501) Overview

- The **LISTAGG** function is used to aggregate a set of string values into a single string
- The values are appended based on the order specified by the WITHIN GROUP clause
- The values may be separated by a separator



LISTAGG - ALL

Output job titles, in ascending order, under the same department according to their job title

```
SELECT WORKDEPT,
       LISTAGG(ALL JOB,', ') WITHIN GROUP (ORDER BY JOB) AS JOB_TITLES FROM DSN8C10.EMP
GROUP BY WORKDEPT!
```

WORKDEPT	JOB_TITLES
A00	CLERK , CLERK , PRES , SALESREP, SALESREP
B01	MANAGER
C01	ANALYST , ANALYST , ANALYST , MANAGER
D11	DESIGNER, DESIGNER, DESIGNER, DESIGNER, DESIGNER, DESIGNER, DESIGNER, DESIGNER, DESIGNER, DESIGNER, MANAGER
D21	CLERK , CLERK , CLERK , CLERK , CLERK , CLERK , MANAGER
E01	MANAGER
E11	MANAGER , OPERATOR, OPERATOR, OPERATOR, OPERATOR, OPERATOR, OPERATOR
E21	FIELDREP, FIELDREP, FIELDREP, FIELDREP, FIELDREP, MANAGER

LISTAGG - DISTINCT

Output job titles, in ascending order, under the same department according to their job title, **eliminating** any duplicate titles

```
SELECT WORKDEPT,
       LISTAGG(DISTINCT JOB,', ') WITHIN GROUP (ORDER BY JOB) AS JOB_TITLES
FROM DSN8C10.EMP
```

WORKDEPT	JOB_TITLES
A00	CLERK , PRES , SALESREP
B01	MANAGER
C01	ANALYST , MANAGER
D11	DESIGNER, MANAGER
D21	CLERK , MANAGER
E01	MANAGER
E11	MANAGER , OPERATOR,
E21	FIELDREP, MANAGER

Summary

Summary

- Db2 11 and Db2 12 build on SQL function provided in previous releases with the following focus areas
  - Native SQL Routines
    - Db2 11 Added The Array Datatype
    - Db2 12 adds SQL PL Triggers
    - Constants
  - Analytic processing
    - Db2 12 Added additional OLAP Functions
  - SQL Family and Vendor compatibility extensions
    - Extensions needed to run popular applications requested by customers and vendors
  - Temporal and Archive Tables enable modern applications
    - Continue to enhance use cases and remove restrictions


Questions?