Platform: z/OS

New & Cool SQL: Version 8

William Favero

IT Software Specialist
IBM North American Lab Services
DB2 for z/OS
IBM Software Group

Session: G10

Wednesday, May 25, 2005 /12:30 PM - 1:40 PM.

INTERNATIONAL DB2 USERS GROUP

What's new in V8?

- Multi-row FETCH and INSERT
- Scalar Fullselect
- Multiple DISTINCT Clauses
- Dynamic Scrollable Cursors
- INSERT within SELECT Statement
- GET DIAGNOSTICS Statement
- Current Package Path Special Register

Multi-Row FETCH and INSERT

- Benefits .....
  - Enhances usability and power of SQL
  - Facilitates Portability
  - Performance improved by eliminating multiple trips between application and DB engine; for distributed, reduced network traffic
  - Combined with scrollable cursors important for browse applications
- What is it? .....
  - Multi-row Fetch:
    - A single fetch statement can retrieve multiple rows of data from the result table of a query as a rowset
    - A rowset is a group of rows of data that are grouped together and operated on as a set
  - Multi-row Insert:
    - A single SQL statement can insert one or more rows into a table or view
    - Multi-row insert can be implemented as either static or dynamic SQL

Multi-row FETCH topics

- DECLARE CURSOR Statement
- Host Variable Arrays
- FETCH Statement
- POSITIONED UPDATE & DELETE Statement

DECLARE CURSOR example

- Declare C1 as the cursor of a query to retrieve a rowset from the table DEPT.
- The prepared statement is MYCURSOR

```
EXEC SQL
    DECLARE CURSOR C1 CURSOR
    WITH ROWSET POSITIONING
    FOR MYCURSOR;
```

- Rowset positioning specifies whether multiple rows of data can
- be accessed as a rowset on a single FETCH statement -- default is WITHOUT ROWSET POSITIONING

Host Variable Arrays

- Host variable array is an array in which each element of the array contains a value for the same column
- Changes to allow host variable arrays
  - COBOL
  - PL/1
  - C++
  - NOTE: Assembler support is limited to cases where USING DESCRIPTOR is allowed. Assembler pre-compiler does not recognize declaration of host variable arrays. Programmer responsible for allocating storage correctly, etc..
- Can only be referenced in multi-row fetch or insert
- In general, arrays may not be arrays of structures

COBOL

Example 1: Declare CURSOR C1 and fetch 10 rows using multi-row FETCH

```
01 OUTPUT-VARS.
     05 NAME OCCURS 10 TIMES.
          49 NAME-LE PIC S9(4)COMP-4 SY C.
          49 NAME-DATA PIC X(40).
     05 SERIAL-NUMBER PIC S9(9)COMP-4 OCCURS 10 TIMES.

PROCEDURE DIVISION.

EXEC SQL
  DECLARE C1 CURSOR WITH ROWSET POSITIONING FOR
  SELECT NAME, SERIAL# FROM CORPORATE.EMPLOYEE  END-EXEC.

EXEC SQL
  OPEN C1  END-EXEC.

EXEC SQL
  FETCH FIRST ROWSET FROM C1 FOR 10 ROWS INTO :NAME,
     :SERIAL-NUMBER  END-EXEC.
```

FETCH example

EXAMPLE 1:

Fetch the previous rowset and have the cursor positioned on that rowset

```
EXEC SQL
    FETCH PRIOR ROWSET FROM C1 FOR 3 ROWS INTO...
```

-- OR --

```
EXEC SQL
     FETCH ROWSET
        STARTING AT RELATIVE -3 FROM C1 FOR 3 ROWS INTO...
```

EXAMPLE 2:

Fetch 3 rows starting with row 20 regardless of the current position of the cursor

```
EXEC SQL
      FETCH ROWSET STARTING AT ABSOLUTE 20
           FROM C1 FOR 3 ROWS INTO...
```

ROWSETs

- A group of rows for the result table of a query which are returned by a single FETCH statement
- Program controls how many rows are returned (i.e., size of the rowset)
  - Can be specified on the FETCH statement (maximum rowset size is 32767)
- Each group of rows are operated on as a rowset
- Ability to intertwine single row and multiple row fetches for a multi-fetch cursor

```
FETCH FIRST ROWSET STARTING AT ABSOLUTE 10
      FROM CURS1
      FOR 6 ROWS INTO :hva1, :hva2;
```

Determining rowset size

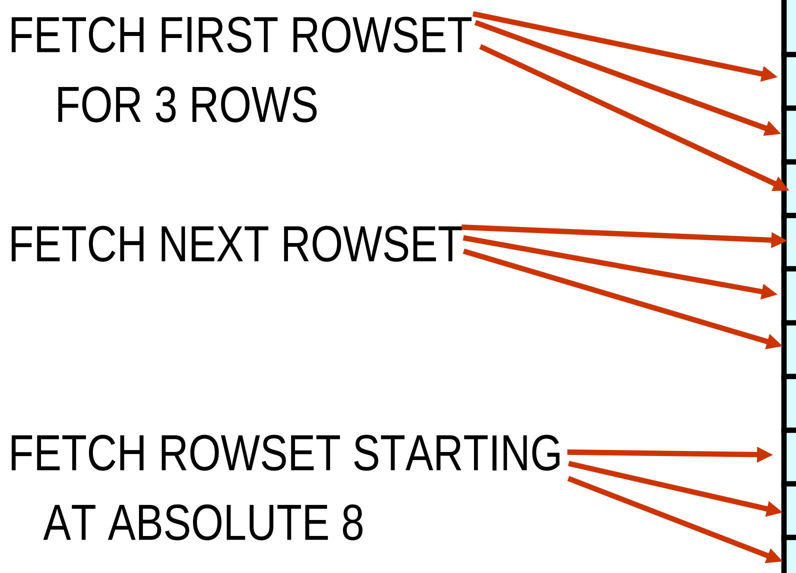
- If FOR n ROWS is NOT specified and cursor is declared for rowset positioning..
- Size of rowset will be the same as the previous rowset fetch as long as
  - It was the previous fetch for this cursor
  - Or the previous fetch was a FETCH BEFORE or FETCH AFTER and the fetch before that was a rowset fetch
- Else rowset is 1

```
FETCH FIRST ROWSET FOR 5 ROWS       Returns 5 rows
FETCH NEXT ROWSET                   Returns the next 5 rows
FETCH NEXT                          Returns a single row
FETCH NEXT ROWSET                   Return a single row
FETCH NEXT ROWSET FOR 3 ROWS        Returns 3 rows
FETCH BEFORE                        Returns 0 rows
FETCH NEXT ROWSET                   Returns 3 rows
```

Fetching beyond the result set

- If you try to fetch beyond the result set you will receive end of data condition
  - i.e., When there are only 5 rows left in result table and you request FETCH NEXT ROWSET FOR 10 ROWS, 5 rows will be returned with an SQLCODE +100
  -
- This includes where FETCH FIRST n ROWS ONLY has been specified

Cursor Positioning : Rowset positioned fetches

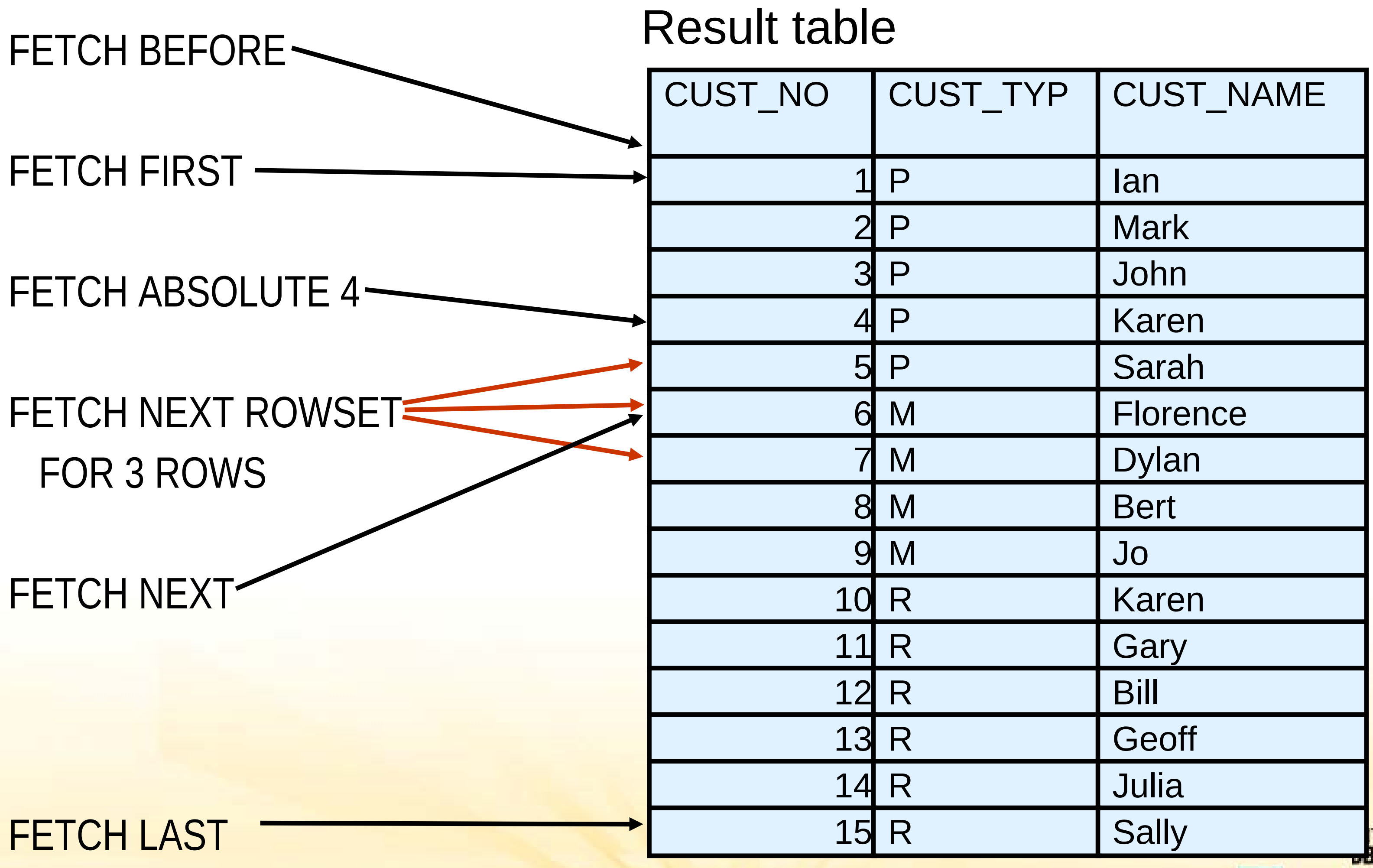


Result table

CUST_NO	CUST_TYP	CUST_NAME
1	P	Ian
2	P	Mark
3	P	John
4	P	Karen
5	P	Sarah
6	M	Florence
7	M	Dylan
8	M	Bert
9	M	Jo
10	R	Karen
11	R	Gary
12	R	Bill
13	R	Geoff
14	R	Julia
15	R	Sally

Note : Cursor is positioned on ALL rows in current rowset

Cursor Positioning : Row positioned fetches



Result table

CUST_NO	CUST_TYP	CUST_NAME
1	P	Ian
2	P	Mark
3	P	John
4	P	Karen
5	P	Sarah
6	M	Florence
7	M	Dylan
8	M	Bert
9	M	Jo
10	R	Karen
11	R	Gary
12	R	Bill
13	R	Geoff
14	R	Julia
15	R	Sally

Positioned UPDATE of Multi-row FETCH

- Allows positioned UPDATE or DELETE to be used on a "wide" cursor

```
UPDATE T1 SET COL1='ABC'
     FOR CURSOR C1
     FOR ROW :hv OF ROWSET;
```

Careful

Multi-row insert

- New third form of insert
  - ► INSERT via VALUES is used to insert a single row into the table or view using values provided or referenced
  - ► INSERT via SELECT is used to insert one or more rows into table or view using values from other tables or views
  - ► INSERT via FOR "n" ROWS form is used to insert multiple rows into table or view using values provided in host variable array
- FOR "n" ROWS
  - ► For static, valid to specify FOR "n" ROWS on INSERT statement (for dynamic INSERT, specify FOR "n" ROWS on EXECUTE statement)
  - ► Maximum value of n is 32767
  - ► Input provided with host variable array -- each array represents cells for multiple rows of a single column
- VALUES clause allows specification of multiple rows of data
  - ► Host variable arrays used to provide values for a column on INSERT
  - ► Example: VALUES (:hva1, :hva2)

Multi-row insert - ATOMIC vs. NOT ATOMIC

- ATOMIC (default) -- if the insert for any row fails, all changes made to database by any inserts are undone
- NOT ATOMIC -- inserts are processed independently
  - ►If errors occur during execution of INSERT, processing continues
  - ►Diagnostics are available for each failed row through GET DIAGNOSTICS
  - ►SQLCODE will indicate if all failed, all were successful or at least one failed

GET DIAGNOSTICS statement

- Enables more diagnostic information to be returned than can be contained in SQLCA
- Returns SQL error information
  - for overall statement
  - for each condition (when multiple conditions occur)
- Supports SQL error message tokens greater than 70 bytes (SQLDA Limitation)

```
INSERT INTO T1 FOR 5 ROWS VALUES (:array);
  GET DIAGNOSTICS :errcount = NUMBER;
       DO || = 1 TO ERR_COUNT;
          GET DIAGNOSTICS FOR CONDITION :||
               :rc = RETURNED_SQLSTATE;
       END;
```

Moved from end of presentation

Some examples for GET DIAGNOSTICS

- To determine how many rows were updated in an UPDATE statement
  - GET DIAGNOSTICS :rcount = ROW_COUNT;
- To handle multiple SQL errors during a NOT ATOMIC multi-row insert
  - GET DIAGNOSTICS :numerrors = NUMBER;
  - Then code a loop to execute the following for the number of errors
    - GET DIAGNOSTICS CONDITION :i :retstate = RETURNED_SQLSTATE
- To see all diagnostic information for an SQL statement
  - GET DIAGNOSTICS :diags = ALL STATEMENT
  - Sample output in :diags
    - Number=1; Returned_SQLSTATE=02000; DB2_RETURNED_SQLCODE=+100;
    - Would continue for all applicable items and for all conditions
    - Items are delimited by semicolons

Scalar Fullselect

- Benefits:
  - Enhances usability and power of SQL
  - Facilitates Portability
  - Conforms with SQL Standards
- What is it?
  - Scalar fullselect is a fullselect, in parentheses, that returns single value
  - Scalar fullselect can be used in an expression
- Example:

```
SELECT PRODUCT, PRICE  FROM PRODUCTS
WHERE PRICE <= 0.7 * (SELECT AVG(PRICE) FROM PRODUCTS);
```

Scalar Fullselect -- Restrictions

- Scalar Fullselect Not Supported in...
  - a CHECK constraint
  - a grouping expression
  - a view that has a WITH CHECK OPTION
  - a CREATE FUNCTION (SQL scalar)
  - a column function
  - ORDER BY clause
  - join-condition of the ON clause for INNER/OUTER JOINS

Multiple DISTINCT clauses

- Benefits .....
  - ►Enhances usability and power of SQL
  - ►DB2 Family Compatibility
- What is it? .....
  - ►Allows more than one DISTINCT keyword on the SELECT or HAVING clause for a query
  - ►DB2 can now evaluate standard deviation & variance column functions

Multiple DISTINCT clauses

- Prior to Version 8 .....
  - SELECT DISTINCT C1, C2 FROM T1;
  - SELECT COUNT(DISTINCT C1) FROM T1;
  - SELECT C1, COUNT(DISTINCT C2) FROM T1 GROUP BY C1;

✓

- With Version 8 .....
  - SELECT DISTINCT COUNT(DISTINCT C1), SUM(DISTINCTC2) FROM T1;
  - SELECT COUNT(DISTINCT C1), AVG(DISTINCT C2) FROM T1 GROUP BY C1;
  - SELECT SUM(DISTINCT C1), COUNT(DISTINCT C1), AVG(DISTINCT C2) FROM T1; GROUP BY C1 HAVING SUM(DISTINCT C1) = 1;

- Not Supported in Version 8 .....
  - SELECT COUNT(**DISTINCT A1,A2**) FROM T1 GROUP BY A2;
  - SELECT COUNT(**DISTINCT(A1,A2)**) FROM T1 GROUP BY A2;

✗

Dynamic Scrollable Cursors

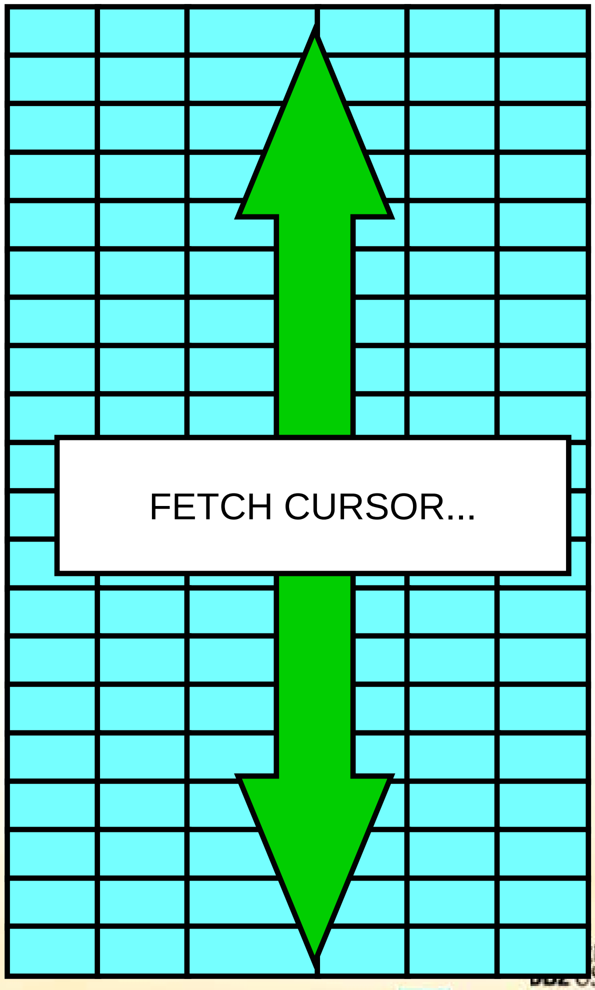
- Benefits:
  - ►Enhances usability and power of SQL
  - ►Facilitates Portability
  - ►Conforms to SQL Standards
  - ►Performance improved by sort elimination
  - ►Elimination of work file (temporary table)
  - ►Scrolls directly on base table (TS scan, IX scan & DPSI)
  - ► No result table materialized at any time
  - ► Immediate visibility of committed updates, inserts, deletes
  - ►Positioned updates and deletes allowed when possible
  - ►Avoid sort by using backward index scan for ORDER BY

Static Scrollable Cursors -- Version 7 Review

- Cursors can be scrolled
  - –backwards
  - –forwards
  - –to an absolute position
  - –to a position relative to the current cursor
  - –before/after position
  - –Sensitive and Insensitive Cursors

- Result table in TEMP DB

- Rows refreshed on demand



Dynamic Scrollable Cursors –V8 new function

- Scrollable cursor that provides access to base table rather than workfile -- allows immediate visibility of updates, deletes, and ***inserts***
- Supported for index and tablespace scan access paths (type 2 indexes)

```
DECLARE C1 SENSITIVE DYNAMIC SCROLL CURSOR
FOR SELECT C1, C2 FROM T1;
```

New DECLARE CURSOR statement attributes

- ASENSITIVE
  - ►DB2 determines sensitivity of cursor
  - ►If read-only...
    - –It behaves as an insensitive cursor
    - –Cursor is INSENSITIVE if SELECT statement does not allow it to be SENSITIVE (Union, Union All, For Fetch Only, For Read Only)
  - ►If not read only, SENSITIVE DYNAMIC is used for maximum sensitivity
  - ►Mainly for Client applications that do not care whether or not the server supports the sensitivity or scrollability
- SENSITIVE DYNAMIC
  - ►Specifies that size of result table is not fixed at OPEN cursor time
  - ►Cursor has complete visibility to changes
    - –All committed inserts, updates, deletes by other application processes
    - –All positioned updates and deletes within cursor
    - –All inserts, updates, deletes by same application processes, but outside cursor
  - ►FETCH executed against base table since no temporary result table created

Implications on FETCH

- INSENSITIVE not allowed with FETCH statement
  - ►If the associated cursor is either declared as SENSITIVE DYNAMIC SCROLL or
  - ►If the cursor is declared ASENSITIVE and DB2 chooses the maximum allowable sensitivity of SENSITIVE DYNAMIC SCROLL
- There are no "holes" as there is no temporary result table
  - ►Special case: If FETCH CURRENT or FETCH RELATIVE + 0 requested but row on which cursor is positioned was deleted or updated so that it no longer meets selection criterion (returns +222)
- •Inserts by the application itself are immediately visible - inserts by others are visible after committed
- Order is always maintained
  - ►If current row deleted, the cursor is positioned before the next row of the original location and there is no current row

Cursor position and scrolling

- At OPEN CURSOR, cursor is positioned before first row
- After FETCH, fetched row becomes current row and cursor positioned on current row
- When FETCH reaches end of file, cursor positioned after last row if scroll number positive and before first row if scroll number negative
- Ability to scroll backwards and forwards through result set which is ever changing
  - ► Note that scroll quantity counts new inserts and has no way of counting deleted rows
  - ► Usage example: airline reservation or credit card processing

Some considerations

- Dynamic scrollable cursors are supported with stored procedures
  - SP itself can update via dynamic scrollable cursor but program calling SP is restricted from updating using allocated cursor
- Scalar functions/arithmetic expressions in SELECT list are reevaluated every fetch
- Column functions (AVG, MIN, MAX, etc.) are calculated once at open cursor
  - Functions may not be meaningful because size of result set can change
- Use of non-deterministic function (built-in or UDF) in WHERE clause of select statement or statement name of scrollable cursor can cause misleading results
  - Result of function can vary from one FETCH to subsequent FETCH of same row
- Parallelism is not supported with dynamic scrollable cursors

Backward index scan enabled

- DB2 will now select an ascending index and use a backward scan to avoid the sort for the descending order
- DB2 will use the descending index to avoid the sort and scan the descending index backwards to provide the ascending order
- To be able to use an index for backward scan,
  - Index must be defined on the same columns as ORDER BY and
  - Ordering must be exactly opposite of what is requested in ORDER BY.
  - i.e., If index defined as DATE DESC, TIME ASC, can do:
    - Forward scan for ORDER BY DATE DESC, TIME ASC
    - Backward scan for ORDER BY DATE ASC, TIME DESC
  - But must sort for
    - ORDER BY DATE ASC, TIME ASC or ORDER BY DATE DESC, TIME DESC

INSERT within SELECT statement

- Benefits:
  - Enables user to immediately determine values inserted in tables by DB2 (identity, sequence, defaults, etc.)and before triggers
  - Cuts down on network cost in application programs
  - Cuts down on procedural logic in stored procedures
- What is it?
  - INSERT statement is now allowed in the FROM clause of a:
    - Select statement that is a subselect
    - SELECT INTO statement
  - Users can automatically retrieve column values created by DB2 INSERT in single SELECT statement
    - Identity columns, sequence values
    - User-defined defaults, expressions
    - Columns modified by BEFORE INSERT triggers
    - ROWIDs

Current Package Path Special Register

- Benefits .....
  - Reduce network traffic and improve CPU/elapsed time for application using DRDA from a z/OS requester
  - Allows nested procedure, user-defined function to be implemented without concern for invoker's runtime environment and allows multiple collections to be specified
  - Easier to switch to/from JDBC and SQLJ
- What is it? .....
  - Current Package Path Special Register:
    - Used for package collection resolution
    - Means for application to specify a list of package collections to DB server (similar to PKLIST on BIND PLAN)
    - DB server (rather than application requester) can search through list and find first package that exists with specified package name
  - Control for applications that do not run under a DB2 plan

Package Resolution today

- Given that multiple collections can be used for packages, how is package resolution managed today?...

  - CURRENT PACKAGESET special register
    - Set to single collection id to indicate any package to be invoked belongs to that collection
    - Application must issue SET CURRENT PACKAGESET before each package is invoked if collection for the package is different from previous package

  - BIND PLAN PKLIST
    - Ability to specify list of collection ids for packages for local OS/390 applications that use plans at execution time

On the Border

- Identity Column Enhancements
- Sequence Objects
- Common Table Expressions
- Materialized Query Tables
- UNICODE SQL, Multiple CCSIDs
- XML Publishing

Summary

- A wealth of SQL changes
- More functionality
- Improved performance

Thanks for coming !

New & Cool SQL: Version 8

Session: G10

William Favero

North American Lab Services

DB2 for z/OS

IBM Software Group

wfavero@attglobal.net