DB2 for z/OS Technical Conference

Dynamic SQL Best Practice and Multi-row FETCH and INSERT

Gareth Jones

DB2 for z/OS Development

jonesgth@uk.ibm.com

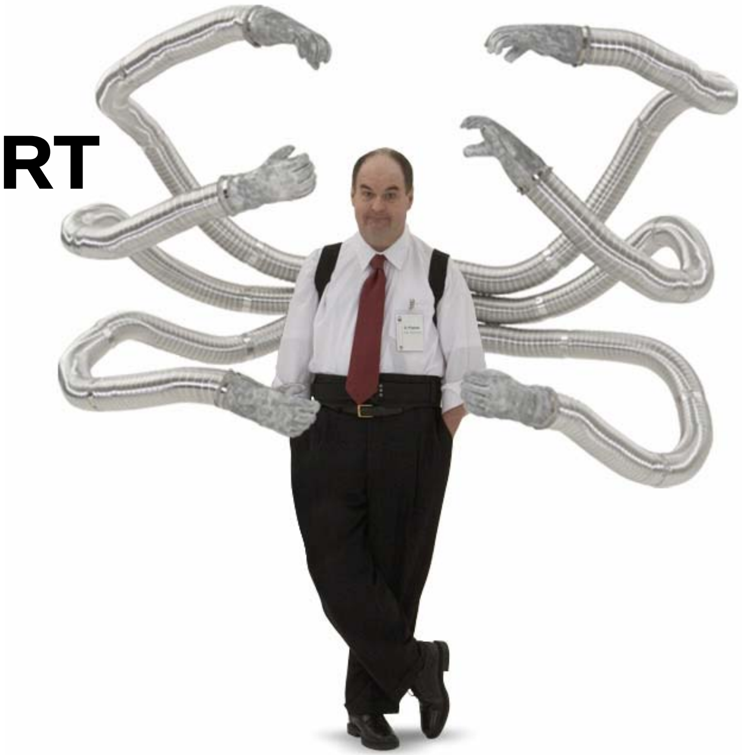
Information Management software

Important Disclaimer

- THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.
- WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.
- IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE.
- IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.
- NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:
  - CREATING ANY WARRANTY OR REPRESENTATION FROM IBM (OR ITS AFFILIATES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS); OR
  - ALTERING THE TERMS AND CONDITIONS OF THE APPLICABLE LICENSE AGREEMENT GOVERNING THE USE OF IBM SOFTWARE.

DB2 for z/OS Technical Conference

Multi-row FETCH and INSERT



IBM Information On Demand 2008

October 26 - 31, 2008 ~ Las Vegas

The Premier Information Management Global Conference

www.ibm.com/events/informationondemand

Information Management software

Multi-row FETCH and INSERT

- Why?
  - 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
- 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

Host Variable Arrays

- Host variable array is an array in which each element of the array contains a value for the same column
  - Changes have been made to allow host variable arrays in:
    - COBOL
    - PL/1
    - C++
    - Limited Assembler support
  - Multi-row operations for Java applications are handled by the JDBC driver and cannot be coded in the application
- Can only be referenced in multi-row fetch or insert
- In general, arrays may not be arrays of structures

COBOL Example

Declare a CURSOR C1 and fetch 10 rows using a multi-row FETCH statement

```
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.
```

C++ Example

Declare an integer and varying character array to hold columns retrieved from a multi-row fetch statement

```
long serial_num(10);
        struct {
                short len;
                char data [18];
        }name [10];
...
EXEC SQL
 DECLARE C1 CURSOR FOR SELECT NAME, SERIAL#
 FROM CORPDATA.EMPLOYEE WITH ROWSET POSITIONING;
...
EXEC SQL OPEN C1;
EXEC SQL
 FETCH FIRST ROWSET FORM C1 FOR 10 ROWS INTO :NAME,
  :SERIAL_NUM;
```

Multiple 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 VALUES... 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, specify FOR "n" ROWS on INSERT statement (for dynamic INSERT, you may also specify FOR "n" ROWS on EXECUTE statement)
  - Maximum value of n is 32767 specified as host-variable, parameter marker, or literal value
  - Input provided with host variable array -- each array represents cells for multiple rows of a single column
  - VALUES... FOR “n” ROWS clause allows specification of multiple rows of data
- Host variable arrays used to provide values for a column on INSERT
  - Example: VALUES (:hva1, :hva2) FOR 10 ROWS

ATOMIC vs NOT ATOMIC

- ATOMIC
  - Traditional behaviour
  - All rows being inserted must successfully be inserted
- NOT ATOMIC CONTINUE ON SQLEXCEPTION
  - Insert rows that are successful
  - Reject rows that are not successful
  - GET DIAGNOSTICS can be used to determine which rows were not successful
  - SQLCODE will indicate if all failed, all were successful or at least one failed

```
EXEC SQL INSERT INTO T1 VALUES (:hva :hvind)
FOR :hv ROWS ATOMIC;
```

- In this example, :hva represents the host variable array and :hvind represents the array of indicator variables

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 intermix row positioned and rowset positioned fetches when a cursor is declared WITH ROWSET POSITIONING

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

Multiple Row FETCH – coding DECLARE CURSOR

- Declare C1 as the cursor of a query to retrieve a rowset from the table DEPT.

```
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

FETCH Examples

- 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...
```

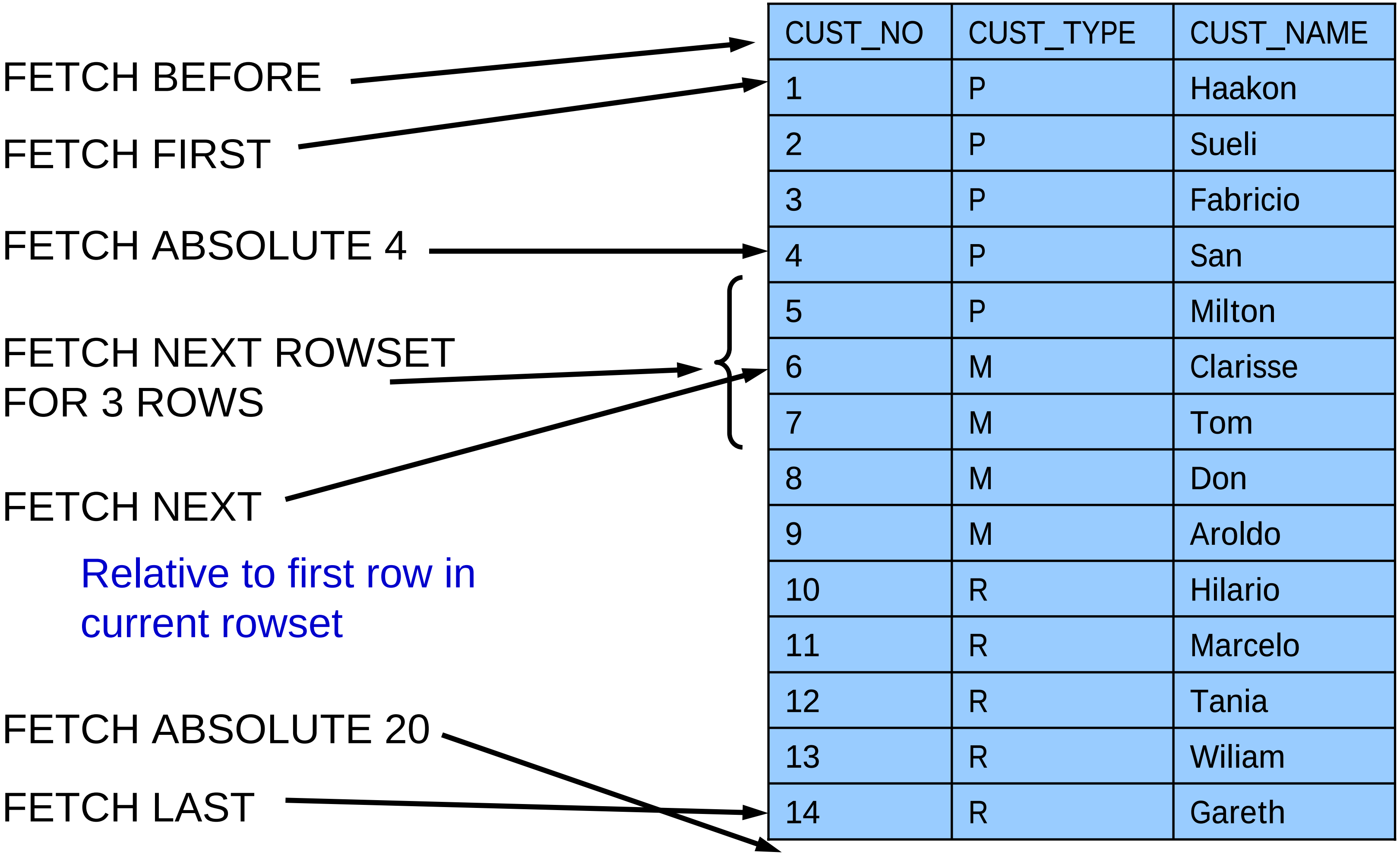
  - -- OR –

```
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...
```

Row and Rowset Positioned Fetches



FETCH BEFORE

FETCH FIRST

FETCH ABSOLUTE 4

FETCH NEXT ROWSET
FOR 3 ROWS

FETCH NEXT

Relative to first row in current rowset

FETCH ABSOLUTE 20

FETCH LAST

CUST_NO	CUST_TYPE	CUST_NAME
1	P	Haakon
2	P	Sueli
3	P	Fabricio
4	P	San
5	P	Milton
6	M	Clarisse
7	M	Tom
8	M	Don
9	M	Aroldo
10	R	Hilario
11	R	Marcelo
12	R	Tania
13	R	Wiliam
14	R	Gareth

Partial Results Sets

- If you fetch beyond the end of the result set, you will receive an 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 - SQLCODE +100
  - SQLERRD(3) will contain the number or rows returned
  - This includes where FETCH FIRST n ROWS ONLY has been specified
- If you fetch beyond the beginning of the result set, you will receive an end of data condition
  - i.e., if you are positioned on rows 3,4,5,6, and 7, and you request FETCH PRIOR ROWSET FOR 10 ROWS, 2 rows will be returned (Rows 1 and 2) - SQLCODE +20237
  - SQLERRD(3) will contain the number or rows returned

Fetching Outside the Result Set – Absolute or Relative

- If you fetch beyond the end of the result set, or beyond the beginning of the result set, you will receive an end of data condition
  - Assume you are positioned on row 5 in a result set with 10 rows.
    - FETCH ROWSET STARTING AT ABSOLUTE 15
    - FETCH ROWSET STARTING AT RELATIVE -7
  - No rows will be returned - SQLCODE +100
  - SQLERRD(3) will contain 0
  - Cursor position will be either “BEFORE” or “AFTER” depending on the direction of the FETCH.

Positioned DELETE

- Assuming cursor CS1 is positioned on a rowset consisting of 10 rows of table T1:
  - The following DELETE statement could be used to DELETE all 10 rows in the rowset

```
EXEC SQL DELETE FROM T1
WHERE CURRENT OF CS1;
```

  - The following DELETE statement could be used to DELETE the 4th row of the rowset.

```
EXEC SQL DELETE FROM T1
WHERE CURRENT OF CS1
FOR ROW 4 OF ROWSET;
```

Positioned UPDATE

- Assuming cursor CS1 is positioned on a rowset consisting of 10 rows of table T1, the following UPDATE statement could be used to update all 10 rows in the rowset

```
EXEC SQL UPDATE T1
SET C1 = 5
WHERE CURRENT OF CS1;
```

- The following is an example of a positioned UPDATE on a rowset cursor

```
UPDATE T1 SET
COL1='ABC‘
WHERE CURRENT OF CS1
FOR ROW :hv OF ROWSET;
```

GET DIAGNOSTICS

- 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 VALUES (:array) FOR 5 ROWS ;
GET DIAGNOSTICS :ERR_COUNT = NUMBER;
  DO i = 1 TO ERR_COUNT;
    GET DIAGNOSTICS FOR CONDITION :i
    :rc = RETURNED_SQLCODE;
  END;
```

- To determine how many rows were updated in an UPDATE statement:

```
GET DIAGNOSTICS :rcount = ROW_COUNT;
```

GET DIAGNOSTICS C++ Example

- In an application, use GET DIAGNOSTICS to handle multiple SQL Errors.

```
long numerrors, counter;
char retsqlstate [5 ];

EXEC SQL GET DIAGNOSTICS :numerrors = NUMBER;
for (i=1;i < numerrors;i++)
  {
  EXEC SQL
    GET DIAGNOSTICS CONDITION :i
    :retsqlstate = RETURNED_SQLSTATE;
  printf("SQLSTATE =%s",retsqlstate);
  }
```

- Execution of this code segment will set and print retsqlstate with the SQLSTATE for each error that was encountered in the previous SQL statement.

DB2 for z/OS Technical Conference

DYNAMIC SQL Usage

IBM Information On Demand 2008

October 26 - 31, 2008 ~ Las Vegas

The Premier Information Management Global Conference

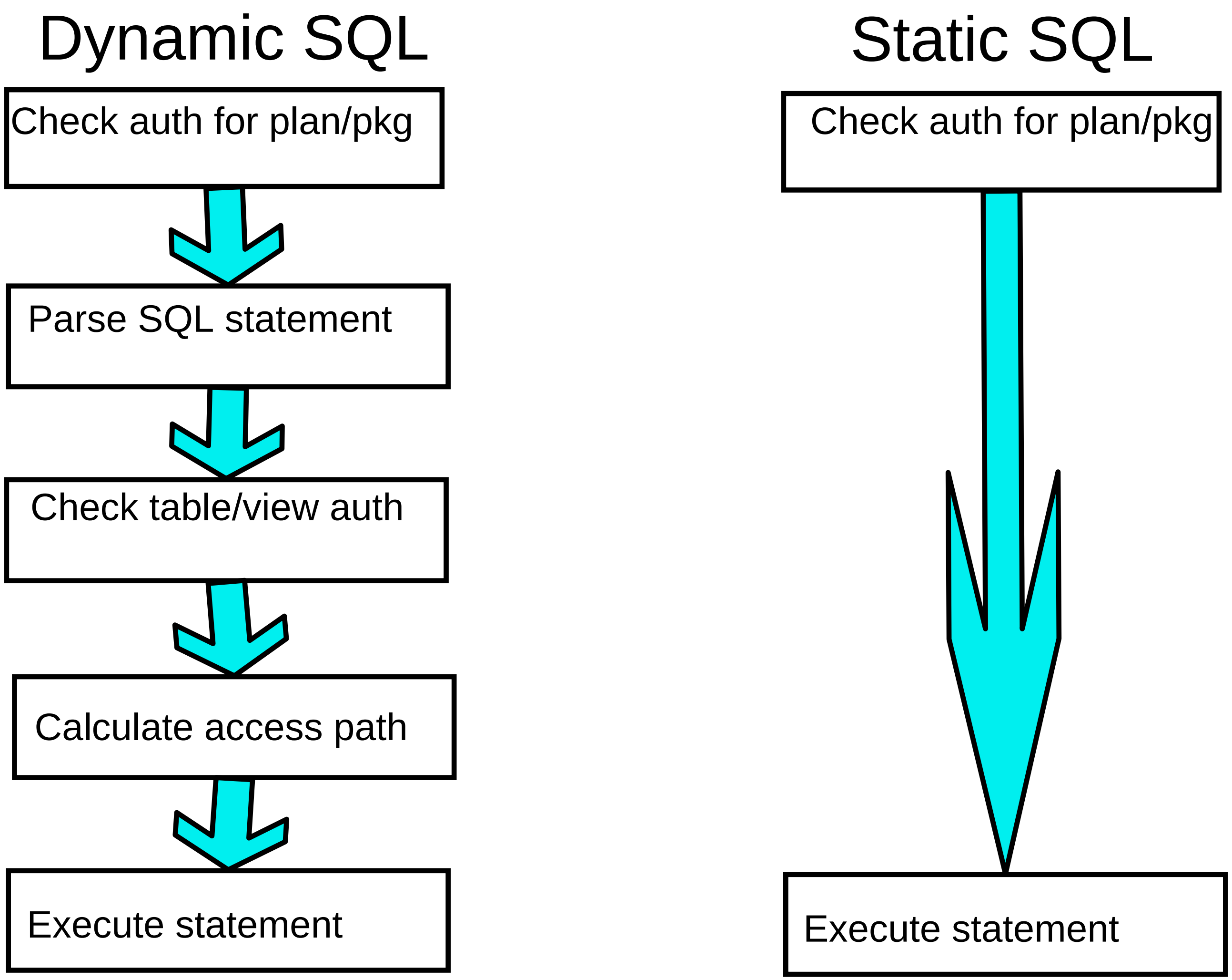
www.ibm.com/events/informationondemand

Information Management software

Static SQL Compared to Dynamic SQL

	Dynamic SQL	Static SQL
Performance	Can approach static SQL performance with help from dynamic SQL caches. Cache misses are costly!	All SQL parsing, catalog access, done at BIND time. Fully optimized during execution.
Access Path Reliability	Unpredictable – Any prepare can get a new access path as statistics or host variables change	Guaranteed – locked in at BIND time All SQL available ahead of time for analysis by EXPLAIN.
Authorization	Privileges handled at object level. All users or groups must have direct table privileges – Security exposure, and administrative burden	Privileges are package based. Only administrator needs table access. Users/Groups have execute authority. Prevent non-authorized SQL execution.
Monitoring, Problem Determination	For remote requests, the database view is typically of a generic JDBC or CLI package – no easy way to tell where any SQL statement came from.	Package view of applications makes it simple to track back to the SQL statement location in the application
Capacity Planning, Forecasting	Difficult to summarize performance data at program level.	Package Level Accounting gives program view of workload to aid accurate forecasting.
Tracking Dependent Objects	No record of which objects are referenced by a compiled SQL statement	Object dependencies registered in database catalog

Execution of Dynamic and Static SQL Requests



Dynamic SQL

Check auth for plan/pkg

Parse SQL statement

Check table/view auth

Calculate access path

Execute statement

Static SQL

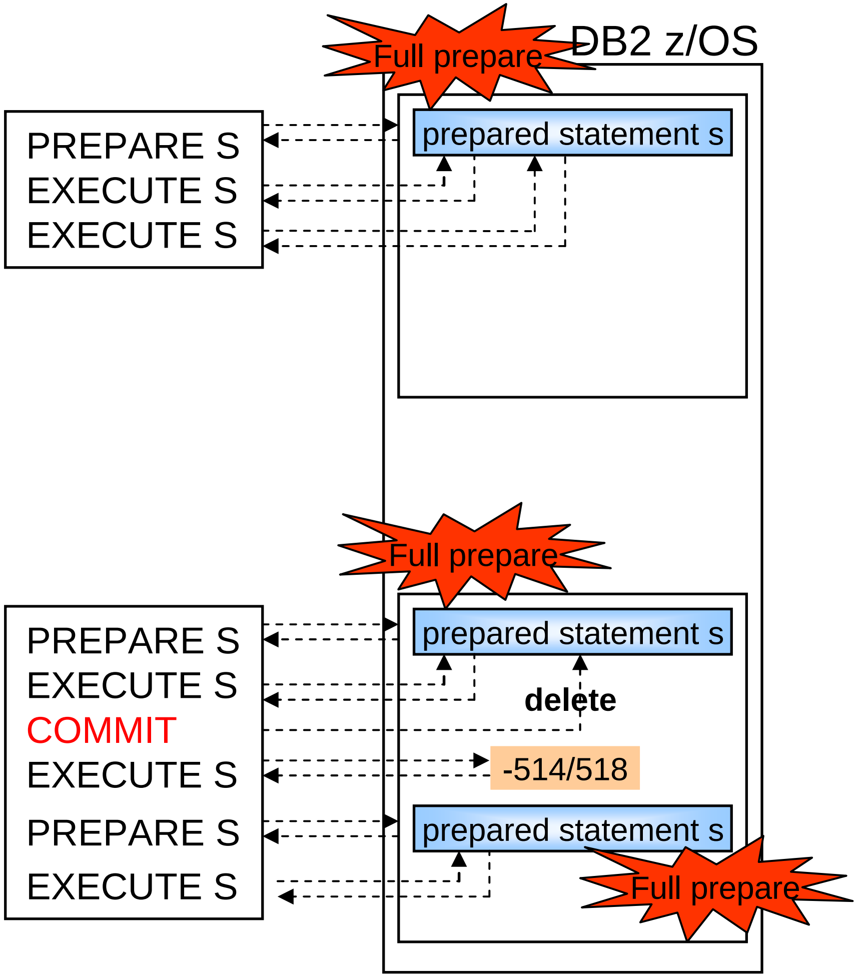
Check auth for plan/pkg

Execute statement

Why Dynamic SQL?

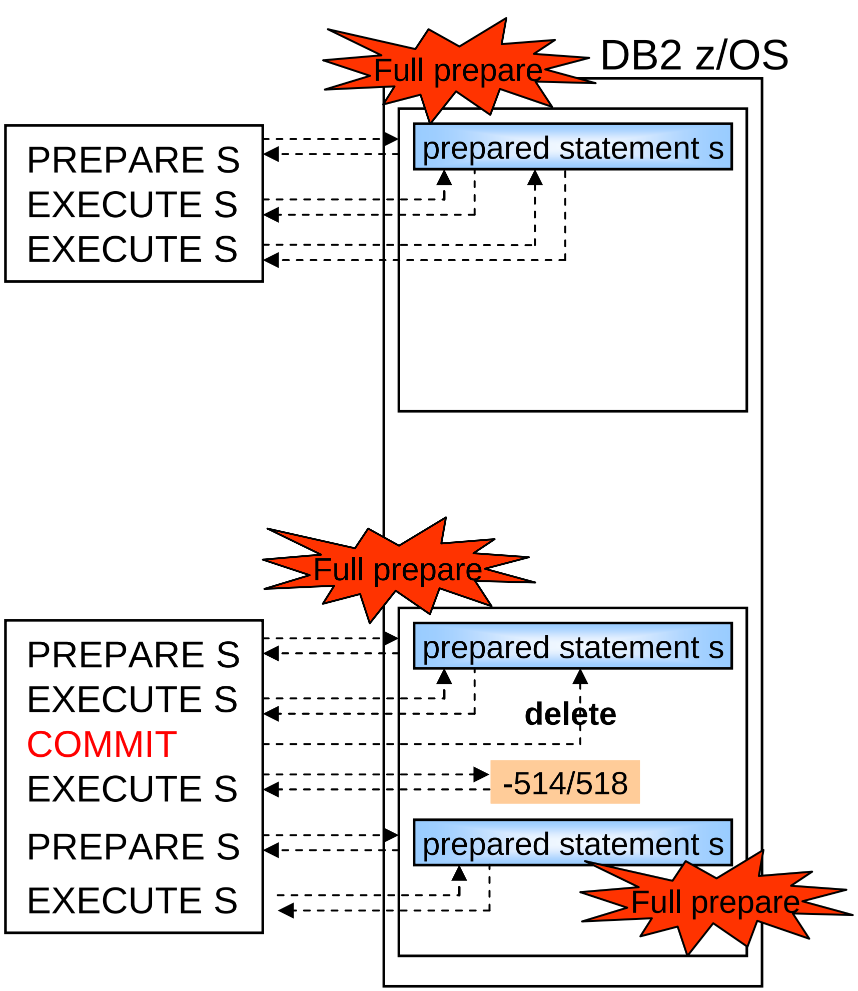
- Alternative way to access DB2 via callable interface:
  - JDBC
  - ODBC
  - Rexx
- Build SQL dynamically to avoid complicated statements with many predicates
- Build SQL dynamically to avoid coding many SQL statements which are executed conditionally based upon program logic.

DB2 Statement Caching - 1

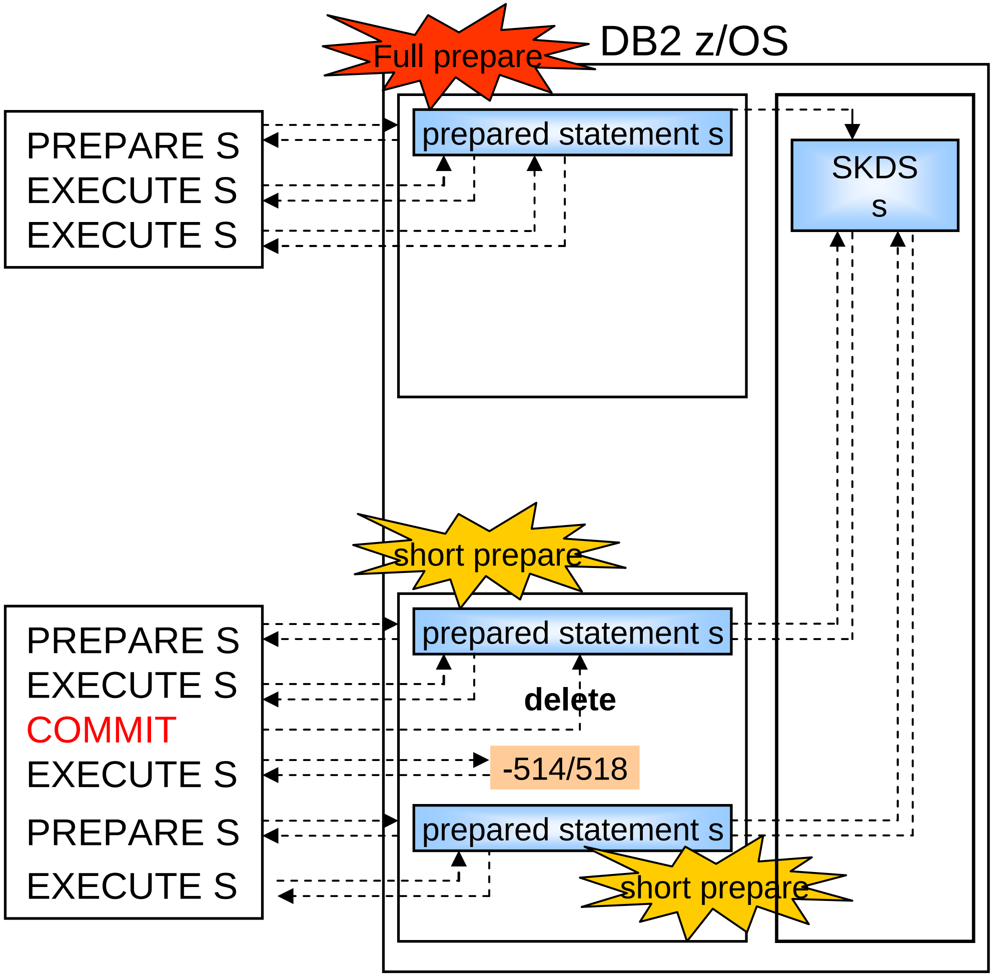


NO CACHING

DB2 Statement Caching - 2



NO CACHING

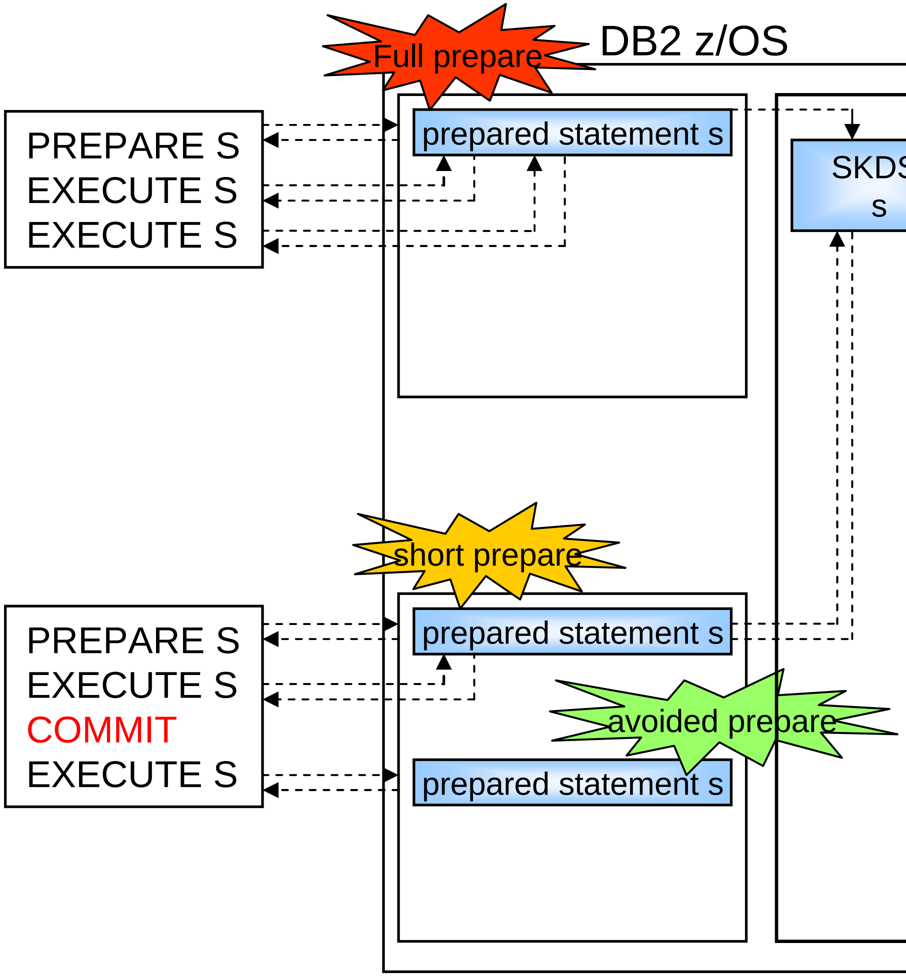


GLOBAL CACHING ONLY

DB2 Statement Caching – Global Caching

- Significant cost to fully prepare a dynamic SQL statement
- Global dynamic statement cache
  - statement text and executable (SKDS) is cached in EDM pool
    - V7 by default in data space
    - V8, V9 above the bar
  - Only first prepare is full prepare, otherwise short prepare, which is a copy from global cache into thread storage
  - No prepared statement is kept in thread storage across commit
- Should be turned on if dynamic SQL is executed in the DB2 system
- Best trade-off between storage and CPU consumption for applications executing dynamic SQL

DB2 Statement Caching - 3



GLOBAL AND LOCAL CACHING

DB2 Statement Caching - Global and Local Caching

- Only first prepare is full prepare, otherwise short prepares
- Prepared statements kept in thread storage across commit (avoided prepares)
    - Same prepared sql statement can be stored in several threads
    - MAXKEEPD limits the stored executable only, the statement text is always stored in thread storage
    - application logic needs to reflect the bind option
- Should only be used selectively for application with a limited number of SQL statements that are executed very frequently
- Should NOT be used for DB2 systems that are constrained in DBM1 storage

Dynamic Statement Cache Controls

- Global Dynamic Statement Cache
  - CACHEDYN=YES (ZPARM)
  - EDMSTMTC= ... ZPARM – size in KB of Global Statement Cache above bar
- Local Dynamic Statement Cache
  - MAXKEEPD= ... ZPARM – maximum number of dynamic statements to keep after commit
    - Global value across DB2 subsystem
  - KEEPDYNAMIC(YES) BIND option
  - CACHEDYN_FREELOCAL= ... ZPARM
    - 0 – DBM1 cannot free cached dynamic statements to relieve DBM1 below-the-bar storage
    - 1 – DBM1 can free.

Dynamic Statement Cache Summary

	CACHEDYN NO	CACHEDYN YES
KEEPDYNAMIC NO	▪ no skeletons cached in EDMP ▪ only full prepares ▪ no prepared statements kept across commits ▪ no statement strings kept across commits	▪ skeletons cached in EDMP ▪ only first prepare full ▪ otherwise short prepares ▪ no prepared statements kept across commits ▪ no statement strings kept across commits
KEEPDYNAMIC YES	▪no skeletons cached in EDMP ▪only full prepares ▪no prepared statements kept across commits ▪statement strings kept across commits – short prepares	▪ skeletons cached in EDMP ▪ only first prepare full, otherwise short prepares ▪ prepared statements kept across commits – avoided prepares ▪ statement strings kept across commits – short prepares

REOPT Enhancement For Dynamic SQL

- We currently have for dynamic SQL
  - REOPT(NONE), REOPT(ONCE) and REOPT(ALWAYS) for dynamic SQL
  - Static only supports REOPT(NONE) and REOPT(ALWAYS)

- V9 ZPARM REOPTEXT = YES / NO
  - NO works as per V8 (default)
  - YES
    - New bind option will be available for REOPT(AUTO)

- REOPTEXT = NO
  - REOPT – NONE, ONCE & ALWAYS
- REOPTEXT = YES
  - REOPT – NONE, ONCE, AUTO & ALWAYS

REOPT - SMART / AUTO

- Ok, so what does it do?
- For dynamic SQL queries with parameter markers
  - DB2 will automatically reoptimize the statement when DB2 detects that the **filtering of one or more of the predicates changes significantly**
  - The newly generated access path will replace the current one and be cached in the statement cache.
- Will reopt at beginning and then monitor runtime values supplied for parameter markers.
  - First optimization is the same as REOPT(ONCE)

Invalidating Statements in the Global Cache

- You may want to invalidate statements in the Global DSC if:
  - An index used by the statement is in RBDP
    - Otherwise index access reverts to relational scan
  - You’ve added a new index to improve access path selection
  - You’ve used OPTHINTS to modify the access path
  - For data collection reasons when monitoring the cache
- V8, V9 use RUNSTATS UPDATE NONE REPORT NO on object accessed by the statement
  - Will invalidate ALL statements accessing that object
  - Will NOT run RUNSTATS, merely performs invalidation

Dynamic Statements and OPTHINTS

- Poorly performing SQL:

```
SELECT *
FROM EMP E, EMPPROJACT EPA
WHERE ...
```

- Add QUERYNO clause and explain

```
EXPLAIN ALL FOR
SELECT *
FROM EMP E, EMPPROJACT EPA
WHERE ...
QUERYNO 729
```

Explain to get access path

Add queryno clause to map
dynamic SQL to specific Queryno.

Checking Plan Table

QUERYNO	METHOD	TNAME	PREF	BINDTIME	OPTHINT
729	0	EMP		2007-12-12 ...	
729	4	EMPROJACT	L	2007-12-12 ...	
729	3			2007-12-12 ...	

- Notice bad join method
  - Compare to previous explain
  - Your analysis indicates hybrid join is inefficient
    - Poor performance

Update Plan Table

QUERYNO	METHOD	TNAME	PREF	BINDTIME	OPTHINT
729	0	EMP		2007-12-12 ...	DYNHINT
729	1	EMPROJACT	L	2007-12-12 ...	DYNHINT
729	3			2007-12-12 ...	DYNHINT

```
UPDATE PLAN_TABLE
SET METHOD = 1
WHERE TNAME =
'EMPPROJACT‘
AND QUERYNO = 729;
```

```
UPDATE PLAN_TABLE
SET OPTHINT =
'DYNHINT'
WHERE QUERYNO = 729
```

TIPS:

1. **Need to set OPTHINT for ALL rows in query block, so use multiple updates!!!**
2. **Double check to ensure access path UPDATES to PLAN_TABLE update only intended rows.**

Use EXPLAIN to Validate the HINT

```
SET CURRENT OPTIMIZATION HINT = 'DYNHINT';
EXPLAIN ALL FOR
SELECT *
FROM EMP E , EMPPROJACCT EPA
WHERE ...
QUERYNO 729;
```

SQL CODE +394 ??

QUERYNO	METHOD	TNAME	PREF	BINDTIME	OPTHINT	HINTUSED
729	0	EMP		2007-12-12 ...	DYNHINT	
729	1	EMPROJACT	L	2007-12-12 ...	DYNHINT	
729	3			2007-12-12 ...	DYNHINT	
729	0	EMP		2007-12-12 ...		DYNHINT
729	1	EMPROJACT	L	2007-12-12 ...		DYNHINT
729	3			2007-12-12 ...		DYNHINT

Hybrid always uses list prefetch, we changed from HYBRID to Nested Loop Join, but didn’t change the prefetch flag…
Let’s be careful out there… (check prefetch, sort flags, etc)

Implementation

```
SET CURRENT OPTIMIZATION HINT = 'DYNHINT';
SELECT *
FROM EMP E , EMPPROJACCT EPA
WHERE ...;
```

Final validation:
SQLCODE = +394, Optimization hint used.
You’ve already used EXPLAIN and PLAN_TABLE to validate how the hint is used.
To be thorough, use PERFORMANCE TRACE CLASS(30) IFCID 22, 63 to see runtime access path

References

- Redbooks at www.redbooks.ibm.com
  - DB2 UDB for z/OS V8 Everything you ever wanted to know… SG24-6079
  - DB2 UDB for z/OS V8 Performance Topics SG24-6465
  - DB2 9 for z/OS Performance Topics SG24-7473
  - Squeezing the Most Out of Dynamic SQL with DB2 for z/OS and OS/390 SG24-6418
- More DB2 for z/OS information at www.ibm.com/software/db2zos
  - E-support (presentations and papers) at www.ibm.com/software/db2zos/support.html

IBM
The Future Runs on System z