Application-enabling features of DB2 10 and 11 for z/OS

The aim of this presentation

- To help ensure that your organization is aware of recently delivered DB2 for z/OS features that can boost agility and productivity with respect to application development

Agenda (1)

- DB2 10 application-enabling features
  - Temporal data support
  - Enhanced SQL user-defined functions
  - RETURN TO CLIENT cursors
  - OLAP moving aggregates
  - LOB enhancements
  - Implicit casting of character string and numeric values
  - Timestamp extensions
  - XML enhancements

Agenda (2)

- DB2 11 application-enabling features
  - Autonomous native SQL procedures
  - Array parameters (and variables) for SQL procedures (and user-defined functions)
  - Temporal special registers and temporal support for views
  - Global variables
  - DB2-managed data archiving
  - New grouping options: GROUPING SETS, ROLLUP, CUBE
  - DB2 integration with Hadoop-managed data
  - XQuery support for XML data

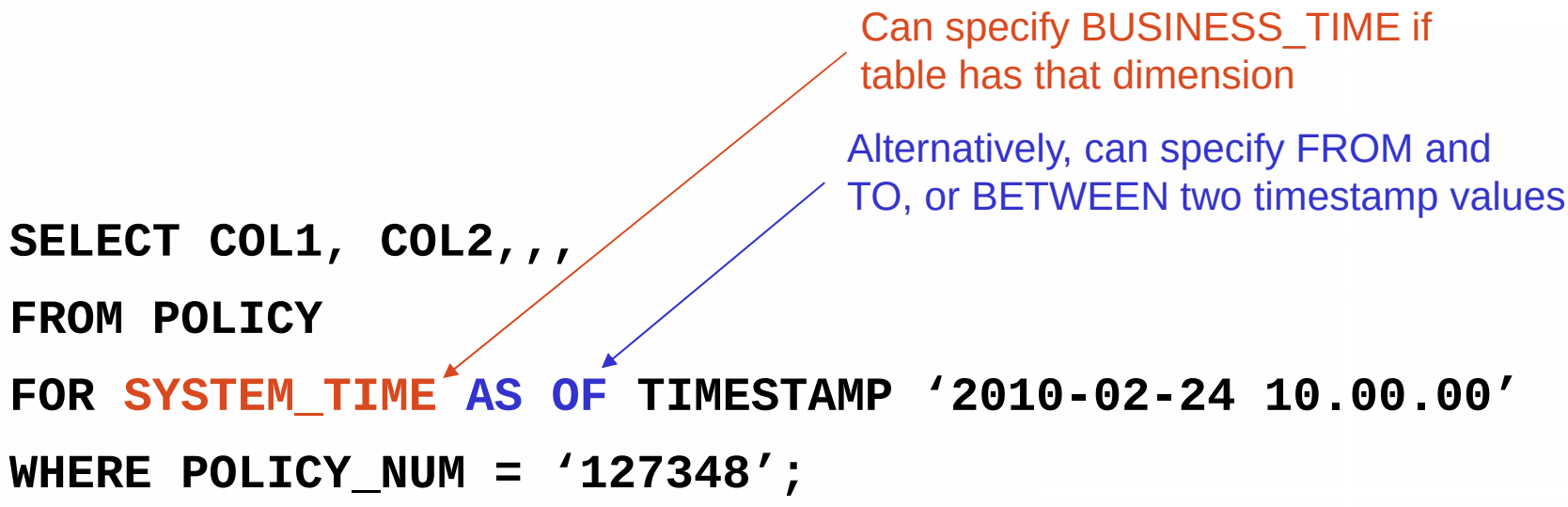
Application enabling features of DB2 10 for z/OS

Temporal data support

- Allows you to give a time dimension to data in a DB2 table
- Two flavors:
  - *System time*: DB2 maintains a history table associated with a base table, and will insert into the history table the “before” version of a row every time a base table row is changed via update or delete
    - DB2 also maintains “from” and “to” timestamp values in base and history table rows, showing when a row in the history table was current, and when a row in the base table became current
  - *Business time*: a dimension that shows when data in a row is valid from a business perspective (e.g., a product price that will go into effect *next year*)
    - You maintain business time values, but DB2 can help by preventing FROM and TO business time period “overlaps” (so one version of a given row will be valid from a business perspective at any given time)
  - You can combine system and business time in one table (“bi-temporal”)

More on temporal data support

- SELECT syntax extended to include the time dimension of a table
- Example: “What was the coverage associated with insurance policy number 127348 at 10 AM on February 24, 2010?”



Advantages of temporal data support

- System time makes it easy to provide an audit history of data changes in a DB2 table
- Business time enables “forward looking” data analysis possibilities
  - Real-world example: forecasting future profit margins using prices that will go into effect at a later time
- DB2-provided temporal capabilities GREATLY increase programmer productivity versus “do it yourself” temporal data functionality
- DB2-implemented temporal table functionality delivers better performance than the do-it-yourself alternative

Enhanced SQL user-defined functions (UDFs)

- Prior to DB2 10, the “logic” in a SQL scalar UDF was restricted to what you could code in the RETURN part of CREATE FUNCTION, and that was quite limited
  - RETURN could not contain a SELECT statement
  - RETURN could not include a column name
- You were basically limited to receiving a value (or values) as input, transforming that value (or values) arithmetically and/or with scalar functions, and returning the result of that transformation
  - Example:

```
CREATE FUNCTION KM_MILES(X DECIMAL(7,2))
RETURNS DECIMAL(7,2)
LANGUAGE SQL
…
RETURN X*0.62;
```

You can still create a UDF like this one, but DB2 10 enabled you to do much more with UDFs written in SQL

Enhanced SQL UDFs (continued)

- Starting with DB2 10, the RETURN part of a SQL scalar UDF can contain a scalar fullselect

```
RETURN(SELECT WORKDEPT FROM EMP WHERE EMPNO = P1);
```

- Also new with DB2 10: the RETURNS part of a SQL scalar UDF can contain a compound SQL statement, in which variables can be declared and which can include logic flow control statements such as IF and WHILE

```
BEGIN
  DECLARE VAR1, VAR2 CHAR(10);
  SET VAR1 = …;
  IF P1 = …;
  RETURN VAR2;
END@
```

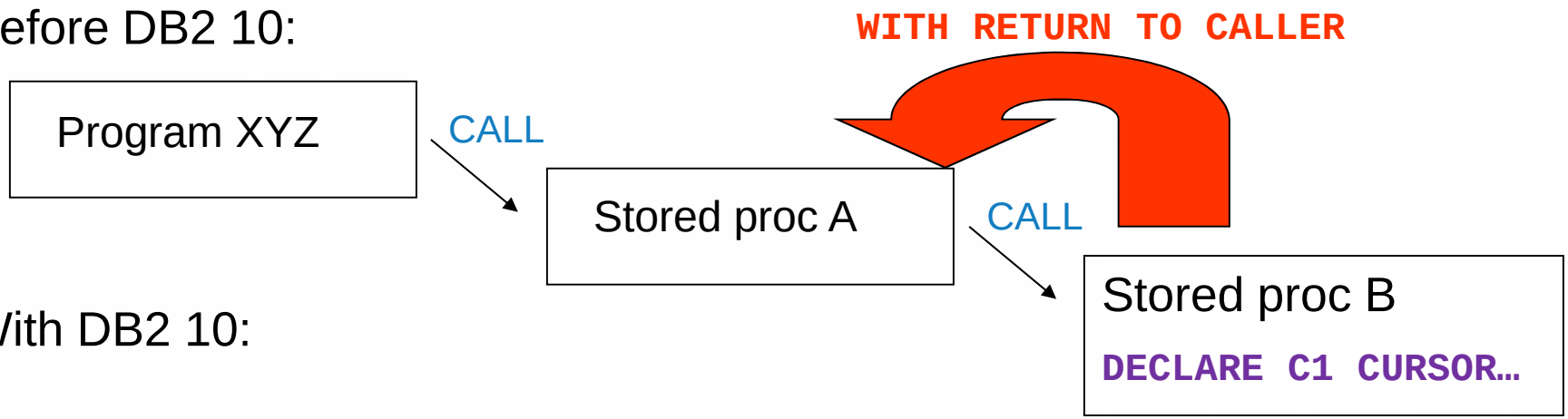
- Also new with DB2 10: SQL table UDFs, which return a result set

RETURN TO CLIENT cursors

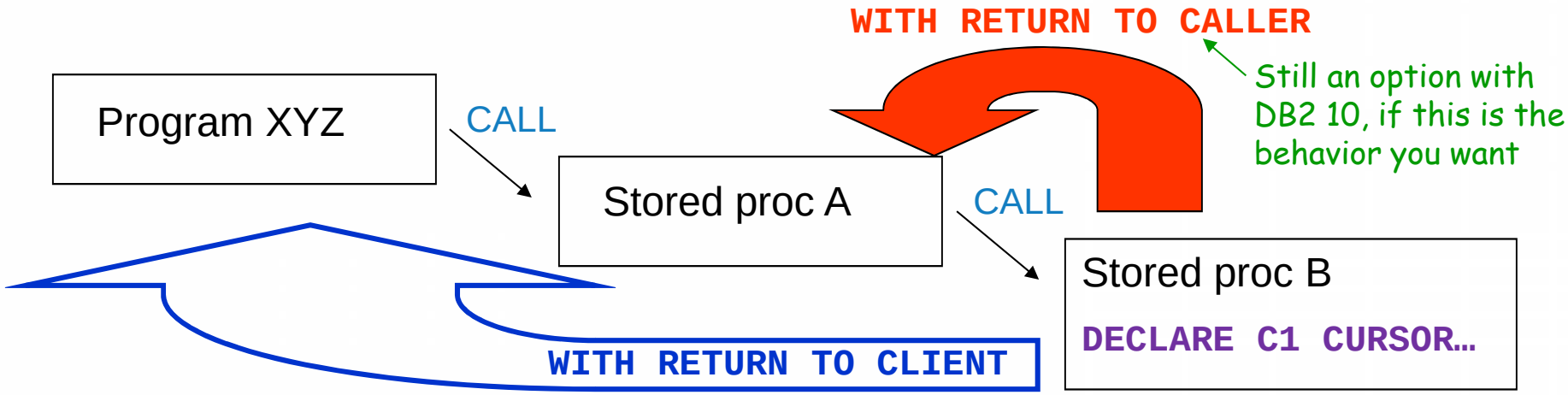
- Prior to DB2 10, a cursor in a stored procedure could be declared WITH RETURN TO CALLER, allowing the result set rows to be directly fetched only by the *direct caller* of the stored procedure
  - Example: program PROG_A calls stored procedure PROC_B, which calls procedure PROC_C, which has a WITH RETURN TO CALLER cursor
  - PROC_B can directly fetch rows from the cursor, but PROG_A cannot
    - If PROG_A needs the result set, PROC_C can put it in a temporary table, and PROG_A can get the rows from that temp table
      - Clunky from a programming perspective, and not optimal for performance
- DB2 10: stored procedure can declare a cursor WITH RETURN TO CLIENT
  - Makes result set rows directly FETCH-able by “top-level” program (i.e., the one that initiated a chain of nested stored procedure calls)

Previous slide's point, in a picture…

- Before DB2 10:



- With DB2 10:



OLAP moving aggregates

- A new (with DB2 10) SQL syntax that allows:
  - Partitioning of a result set (e.g., by name)
  - Ordering of rows within result set partitions (e.g., by date)
  - Generation of aggregate values based on the “moving” current position within a set of rows (e.g., sum of sales for the current row plus the two preceding rows)
  - Example:

```
SELECT NAME, DATE, UNITS_SOLD,
SUM(UNITS_SOLD) OVER(PARTITION BY NAME
                     ORDER BY DATE
                     ROWS BETWEEN 2 PRECEDING AND CURRENT
       ROW) SUM
FROM PRODUCT_SALES;
```

The desired result set partitioning value

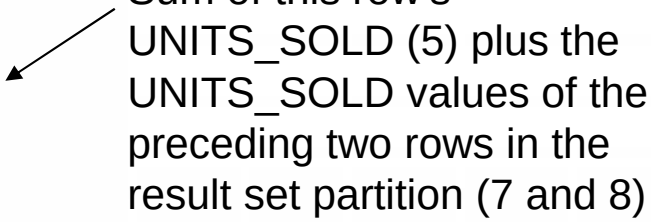
The desired ordering of rows within result set partitions

The desired aggregate function

The desired scope of aggregation as DB2 moves through the result set partitions

The result of the SELECT on the previous slide

NAME	DATE	UNITS_SOLD	SUM
Jones	2015-01-10	7	7
Jones	2015-01-11	8	15
Jones	2015-01-12	5	20
Jones	2015-01-13	6	19
Smith	2015-01-10	4	4
Smith	2015-01-11	9	13
Smith	2015-01-12	8	21
Smith	2015-01-13	5	22



LOB enhancements: inlining

- Prior to DB2 10, every bit of every value in a LOB column had to be physically stored in a separate LOB table space (the LOB values logically appear to be in the base table rows)
- With DB2 10, a LOB column's definition can include a specification of the amount of space in the base table that can be occupied by LOB values
  - The portion (if any) of a value over the limit is stored in LOB table space
- Great for a LOB column for which relatively few values are truly large
  - Can significantly improve the performance of LOB-reading and LOB-inserting programs (and utilities) when most of a LOB column's values can be completely in-lined
  - Also allows creation of index on expression on in-lined portion of a CLOB column (using the SUBSTR)
    - Example: if contracts are stored in a CLOB column, and if data in bytes 10 through 20 is always the contract number, can build index on that

LOB enhancements: utilities

- Variable-block spanned (VBS) record format now supported for data sets used for table UNLOAD and LOAD (referring to SYSREC data set)
  - What this means: you can unload a table with a LOB column (or columns) and have ALL of the data – LOB and non-LOB – go into a single data set
  - And reverse is true for LOAD (i.e., data – LOB and non-LOB values – can be loaded from a single input data set
    - Before DB2 10, had to unload individual LOB values to members of a PDS, or to individual files in the z/OS UNIX System Services file system (and reverse was true for LOAD)
    - DB2 10 spanned record support greatly simplifies use of UNLOAD and LOAD for tables with LOB columns, and substantially boosts performance
- DB2 10 also delivered support for online REORG of LOB table space with SHRLEVEL(CHANGE)

Implicit casting of character, numeric values

- Consider this statement:

```
SELECT 1 CONCAT '+' CONCAT 1 CONCAT '=' CONCAT 2
FROM SYSIBM.SYSDUMMY1;
```

- In a pre-DB2 10 environment, that statement gets this result:

```
SQLCODE = -171, ERROR: THE DATA TYPE, LENGTH,
OR VALUE OF ARGUMENT 1 OF CONCAT IS INVALID
```

- In a DB2 10 (new-function mode) system, you get this:
  - `1+1=2`
- Works assignment (SET) statements, too (but not for special registers)
- Numeric values are implicitly cast to VARCHAR, character values are implicitly cast to DECFLOAT(34)
  - Why? Because VARCHAR and DECFLOAT(34) are compatible with all other character and numeric data types, respectively

Timestamp extensions

- New with DB2 10: timestamp values down to the picosecond (that's a trillionth of a second)
  - One reason this was needed: mainframe engines are so fast now that microsecond-level timestamps (often defined as unique keys in DB2 tables) can regularly produce duplicate values
- Also new with DB2 10: variable-precision timestamps
  - From 0 (no fractions of a second) to 12 (picosecond-level precision), with 6 being the default
  - Syntax: TIMESTAMP(n)
- Another DB2 10 enhancement: TIMESTAMP(n) WITH TIME ZONE
  - New data type
  - Sample value: '2012-10-03-10.15.00.123456-05:00' ← Difference between local time and UTC

XML enhancements

- With DB2 10, you can specify in the definition of a table the XML schema that is to be used to validate data inserted into an XML column
  - No longer have to invoke DB2-supplied user-defined function to accomplish schema validation
  - Additionally, DB2 10 XML schema validation is done “in the DB2 engine”
    - Better performance, and zIIP-eligible
- And, you can update part of an XML document (versus replacing the whole thing) via new XMLMODIFY built-in function
  - Can insert a node into an XML document, replace a node, delete a node, or replace values of a node
- Also, the CHECK DATA utility can check on the structural validity of XML documents in an XML table space
  - Pre-DB2 10: only checked consistency between base table and XML table space

Application enabling features of DB2 11 for z/OS

Autonomous native SQL procedures

- A DB2 11 native SQL procedure can function as an autonomous transaction
  - How it's done: AUTONOMOUS option specified in CREATE PROCEDURE (or ALTER PROCEDURE) statement
    - Specified instead of COMMIT ON RETURN YES/NO
  - An autonomous SQL procedure commits on returning to the calling program, but (unlike the case when COMMIT ON RETURN YES is in effect) that commit does NOT affect the calling program's unit of work
  - An autonomous SQL procedure's unit of work (UOW) is independent of the calling program's UOW – if the calling program's UOW is rolled back, data changes made by autonomous SQL procedure *will not be rolled back*
    - Very useful if you require that a data update be accomplished when a transaction executes, and you need that update to persist *even if the transaction subsequently fails*
  - A restriction: one autonomous SQL procedure can't call another

Array parameters (and variables) for SQL procedures (and UDFs)

- DB2 11: array parameters can be passed to (and/or received from), and array variables can be declared in, native SQL procedures (and the same is true for SQL user-defined functions)
  - Call to SQL procedure with array input or output parameter can come from a SQL PL routine, a Java program, or a .NET program (for latter two, via IBM Data Server Driver type 4 driver)
    - If .NET caller, array must be input parameter
  - An array in this context is a form of a DB2 user-defined data type (UDT) – you create it, then you use it
  - Built-in functions are provided to:
    - Construct arrays
    - Derive tables from arrays
    - Obtain information about arrays
    - Navigate array elements

More on array parameters and variables

- There are two array types:
  - Ordinary
    - Has a user-defined upper bound on number of elements (defaults to INTEGER high value)
    - Elements referenced by their ordinal position in the array
  - Associative
    - No user-defined upper bound on number of elements
    - Elements are ordered by and can be referenced via array index values
    - Values in a given array index are INTEGER or VARCHAR, are unique, and don't have to be contiguous

This is an ordinary array – associative array would have data type of index values (e.g., VARCHAR(8)) after ARRAY keyword

```
CREATE TYPE PHONENUMBERS AS DECIMAL(10,0) ARRAY[50];
```

Data type of values in the array

Max number of elements in ordinary array (defaults to about 2 billion)

Temporal special registers

- The need: what if you want a program (or just a SQL statement) to have an other-than-current view of temporal data, but you don't want to change the program's code?
- Solution: two new special registers delivered with DB2 11
  - CURRENT TEMPORAL SYSTEM_TIME
  - CURRENT TEMPORAL BUSINESS_TIME
- When set to a non-null value, has the effect of adding the following to a SELECT statement that targets a temporal-enabled table (in this case, use of system time is assumed):
  - FOR SYSTEM_TIME AS OF CURRENT TEMPORAL SYSTEM_TIME
- Example of setting special register's value:

```
SET CURRENT TEMPORAL SYSTEM_TIME = CURRENT TIMESTAMP - 1 YEAR;
```

(this would result in a program having a view of data that was current as of one year ago)

More on temporal special registers

- A special register non-null value, once set, remains in effect for that particular session (thread) until it's changed (setting to null has the effect of "turning the special register off")
  - But if set within a routine (stored procedure or UDF), the new value is not passed back to the invoking application
- SYSTIMESENSITIVE, BUSTIMESENSITIVE bind options determine whether or not SQL statements (static or dynamic) issued through a package will be affected by temporal special registers
  - Default value is YES
- If CURRENT TEMPORAL SYSTEM_TIME is set to non-null value for a thread, data modification statements targeting system time-enabled tables are not allowed

Temporal support for views

With DB2 11, you can use temporal predicates when referring to a view defined on a temporal table (but you can't use a temporal predicate in defining a view)

```
CREATE TABLE POLICY_BITEMPORAL
(EMPL VARCHAR(4) NOT NULL,
 TYPE VARCHAR(4),
 PLCY VARCHAR(4) NOT NULL,
 COPAY VARCHAR(4),
 SYS_BEG TIMESTAMP(12) GENERATED ALWAYS AS ROW BEGIN NOT NULL,
 SYS_END TIMESTAMP(12) GENERATED ALWAYS AS ROW END   NOT NULL,
 SYS_TMP TIMESTAMP(12) GENERATED ALWAYS AS TRANSACTION START ID,
 PERIOD SYSTEM_TIME   (SYS_BEG, SYS_END),
 BUS_BEG DATE NOT NULL,
 BUS_END DATE NOT NULL,
 PERIOD BUSINESS_TIME (BUS_BEG, BUS_END),
 PRIMARY KEY (EMPL,PLCY, BUSINESS_TIME WITHOUT OVERLAPS)
);
```

Base table

```
CREATE VIEW VIEW_POLICY_BITEMPORAL_2012_ONLY AS
     SELECT * FROM POLICY_BITEMPORAL
 WHERE BUS_BEG <= '12/31/2012'
   AND BUS_END >= '01/01/2012' WITH CHECK OPTION;
```

```
CREATE VIEW VIEW_POLICY_BITEMPORAL_2012_ONLY AS
 SELECT * FROM POLICY_BITEMPORAL
 FOR BUSINESS_TIME FROM '01/01/2012' TO '12/30/2012';
```

SQLCODE -4736

Temporal predicate

View

Temporal predicate

```
SELECT EMPL,TYPE,PLCY,COPAY,BUS_BEG,BUS_END
 FROM VIEW_POLICY_BITEMPORAL_2012_ONLY
 FOR BUSINESS_TIME AS OF '12/30/2012';
```

Global variables

- The need: how can you pass data values from one SQL statement to another in the context of a thread?
  - Before DB2 11:
    - Do it with application code (values placed into variables by one SQL statement are copied to variables used as input to another SQL statement)
    - Want a trigger to be able to access those values? Not easy…
  - DB2 11: use global variables
- You can create your own global variables using the new CREATE VARIABLE statement
  - DB2 11 also provides a few built-in global variables:
    - SYSIBM.CLIENT_IPADDR
    - SYSIBMADM.GET_ARCHIVE
    - SYSIBMADM.MOVE_TO_ARCHIVE

More on this archive stuff momentarily…

Global variables example

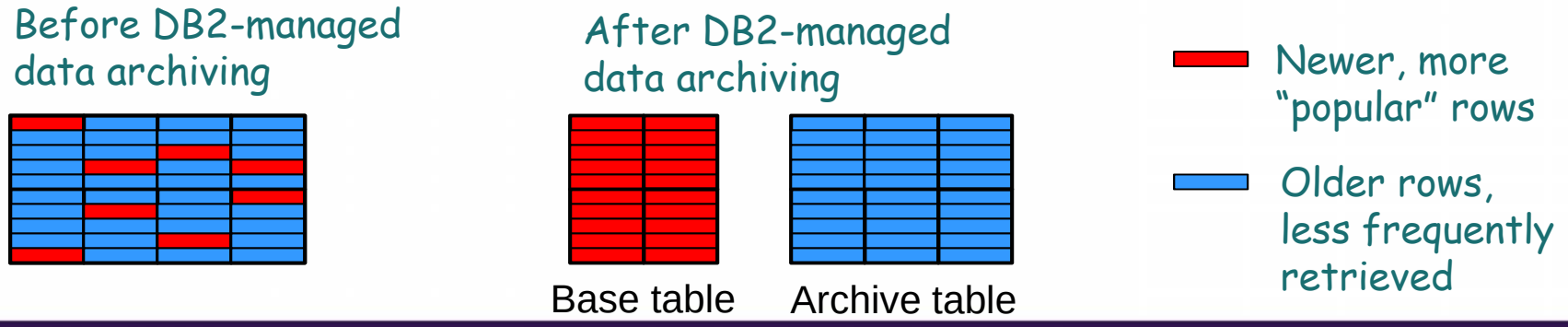
Assign value to a (previously created) global variable

Reference the global variable

```
1> SET CAT_OBJECT = 'SYSTABLES'

1> SELECT SUBSTR(CAT_OBJECT,1,30) AS "CATALOG OBJECT"
2>         ,SUBSTR(CREATOR,1,10) AS CREATOR
3>         ,SUBSTR(TBNAME,1,30) AS "CHILD TABLE"
4>         ,DELETERULE
5>    FROM SYSIBM.SYSRELS
6>    WHERE REFTBCREATOR = 'SYSIBM'
7>      AND REFTBNAME = CAT_OBJECT

 CATALOG OBJECT                 CREATOR    CHILD TABLE                    DELI
 ------------------------------ ---------- ------------------------------ ----
 SYSTABLES                      SYSIBM     SYSCONTROLS                    C
 SYSTABLES                      SYSIBM     SYSTABLES_PROFILES             C
 SYSTABLES                      SYSIBM     SYSCOLUMNS                     C
 SYSTABLES                      SYSIBM     SYSRELS                        C
 SYSTABLES                      SYSIBM     SYSINDEXES                     C
 SYSTABLES                      SYSIBM     SYSTABAUTH                     C
 SYSTABLES                      SYSIBM     SYSSYNONYMS                    C
 SYSTABLES                      SYSIBM     SYSRELS                        C
 SYSTABLES                      SYSIBM     SYSTABSTATS_HIST               C
 SYSTABLES                      SYSIBM     SYSTABLES_HIST                 C
 SYSTABLES                      SYSIBM     SYSTABCONST                    C
 SYSTABLES                      SYSIBM     SYSCONSTDEP                    C
 SYSTABLES                      SYSIBM     SYSTRIGGERS                    C
 SYSTABLES                      SYSIBM     SYSTABSTATS                    C
 SYSTABLES                      SYSIBM     SYSCHECKS                      C
1> SELECT SUBSTR(CAT_OBJECT,1,30) AS "CATALOG OBJECT"
2>         ,SUBSTR(REFTBCREATOR,1,10) AS CREATOR
3>         ,SUBSTR(REFTBNAME,1,30) AS "PARENT TABLE"
4>         ,DELETERULE
5>    FROM SYSIBM.SYSRELS
6>    WHERE CREATOR = 'SYSIBM'
7>      AND TBNAME = CAT_OBJECT

 CATALOG OBJECT                 CREATOR    PARENT TABLE                   DELI
 ------------------------------ ---------- ------------------------------ ----
 SYSTABLES                      SYSIBM     SYSTABLESPACE                  C
```

DB2-managed data archiving

- NOT the same thing as system time temporal data
  - When versioning (system time) is activated for a table, the “before” images of rows made “non-current” by update or delete are inserted into an associated history table
  - With DB2-managed archiving, rows in archive table are current in terms of validity – they’re just older than rows in associated base table
    - The idea: when most access is to rows recently inserted into table, moving older rows to archive table *can improve performance of newer-row retrieval*
    - Particularly helpful when data clustered by non-continuously-ascending key
    - People have long done this themselves – DB2 11 makes it easier



DB2-managed data archiving – how it's done

1. DBA creates table (T1_AR, for example) to be used as archive for table T1
2. DBA tells DB2 to enable archiving for T1, using archive table T1_AR

```
ALTER TABLE T1 ENABLE ARCHIVE USE T1_AR;
```

3. Program deletes to-be-archived rows from T1
   - If program sets built-in global variable SYSIBMADM.MOVE_TO_ARCHIVE to 'Y', *all it has to do is delete from T1* – DB2 will move deleted rows to T1_AR
4. Bind packages appropriately (bind option affects static and dynamic SQL)
   - If a program is to ALWAYS access ONLY the base table, it should be bound with ARCHIVESENSITIVE(NO)
   - If a program is to SOMETIMES or ALWAYS access rows in the base table and the archive table, it should be bound with ARCHIVESENSITIVE(YES)
     - If program sets built-in global variable SYSIBMADM.GET_ARCHIVE to 'Y', and issues SELECT against base table, DB2 will automatically drive that SELECT against associated archive table, too, and will merge results with UNION ALL

New grouping option: GROUPING SETS

Example: determine average total compensation for WorkDept, Job, and EdLevel sets

```
select workdept, job, edlevel,
       decimal(avg(salary+bonus+Comm),9,2) as Compensation
from dsn81110.emp
group by grouping sets
  (workdept, job, edlevel)
;
```

Basically means, "group by each of these columns, in turn"

	WORKDEPT	JOB	EDLEVEL	COMPENSATION
1	A00	NULL	NULL	45038.00
2	B01	NULL	NULL	45350.00
3	C01	NULL	NULL	32725.50
4	D11	NULL	NULL	27667.81
5	D21	NULL	NULL	28221.85
6	E01	NULL	NULL	44189.00
7	E11	NULL	NULL	23115.85
8	E21	NULL	NULL	26497.00
9	NULL	ANALYST	NULL	29597.33
10	NULL	CLERK	NULL	27771.25
11	NULL	DESIGNER	NULL	26891.60
12	NULL	FIELDREP	NULL	26048.00
13	NULL	MANAGER	NULL	38330.57
14	NULL	OPERATOR	NULL	21513.50
15	NULL	PRES	NULL	57970.00
16	NULL	SALESREP	NULL	51420.00
17	NULL	NULL	12	17270.33
18	NULL	NULL	14	27549.42

New grouping option: ROLLUP

Example: determine average total compensation for the various hierarchies of WorkDept, Job, and EdLevel, and for overall set

- Column order in GROUP BY expression affects result set
- ORDER BY helps with readability

```
select workdept, job, edlevel,
       decimal(avg(salary+bonus+Comm),9,2) as Compensation
from dsn81110.emp
group by rollup
  (workdept, job, edlevel)
order by workdept, job, edlevel
;
```

	WORKDEPT	JOB	EDLEVEL	COMPENSATION
1	A00	CLERK	14	32190.00
2	A00	CLERK	NULL	32190.00
3	A00	PRES	18	57970.00
4	A00	PRES	NULL	57970.00
5	A00	SALESREP	18	51720.00
6	A00	SALESREP	19	51120.00
7	A00	SALESREP	NULL	51420.00
8	A00	NULL	NULL	45038.00
9	B01	MANAGER	18	45350.00
10	B01	MANAGER	NULL	45350.00
11	B01	NULL	NULL	45350.00
49	NULL	NULL	NULL	30198.16

You get a grouping by all values of column 1, column 2, and column 3; a grouping by all values of column 1 and column 2; and a grouping by all values of column 1

You also get an aggregate over all qualifying rows

New grouping option: CUBE

- Example: determine average total compensation for various combinations of WorkDept, Job, and EdLevel
  - Column order in GROUP BY expression doesn't matter
  - ORDER BY helps with readability

```
select workdept, job, edlevel,
       decimal(avg(salary+bonus+Comm),9,2) as Compensation
from dsn81110.emp
group by cube
  (workdept, job, edlevel)
order by workdept, job, edlevel
;
```

	WORKDEPT	JOB	EDLEVEL	COMPENSATION
1	A00	CLERK	14	32190.00
2	A00	CLERK	NULL	32190.00
3	A00	PRES	18	57970.00
4	A00	PRES	NULL	57970.00
5	A00	SALESREP	18	51720.00
6	A00	SALESREP	19	51120.00
7	A00	SALESREP	NULL	51420.00
8	A00	NULL	14	32190.00
9	A00	NULL	18	54845.00
10	A00	NULL	19	51120.00
11	A00	NULL	NULL	45038.00
12	B01	MANAGER	18	45350.00
13	B01	MANAGER	NULL	45350.00
14	B01	NULL	18	45350.00

You get grouping by all values of all three columns, by all values of all combinations of two of the three columns, and by all values of each individual column

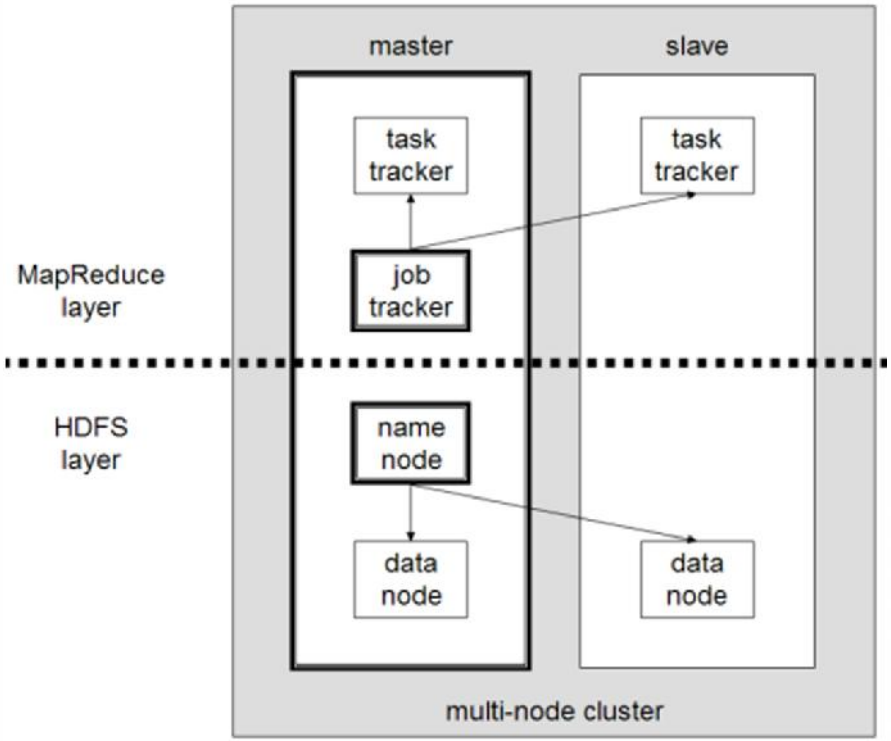
105	NULL	NULL	19	51120.00
106	NULL	NULL	20	42110.00
107	NULL	NULL	NULL	30198.16

You also get an aggregate over all qualifying rows

DB2 integration with Hadoop-managed data

- Hadoop: an open source software framework that supports data-intensive distributed applications

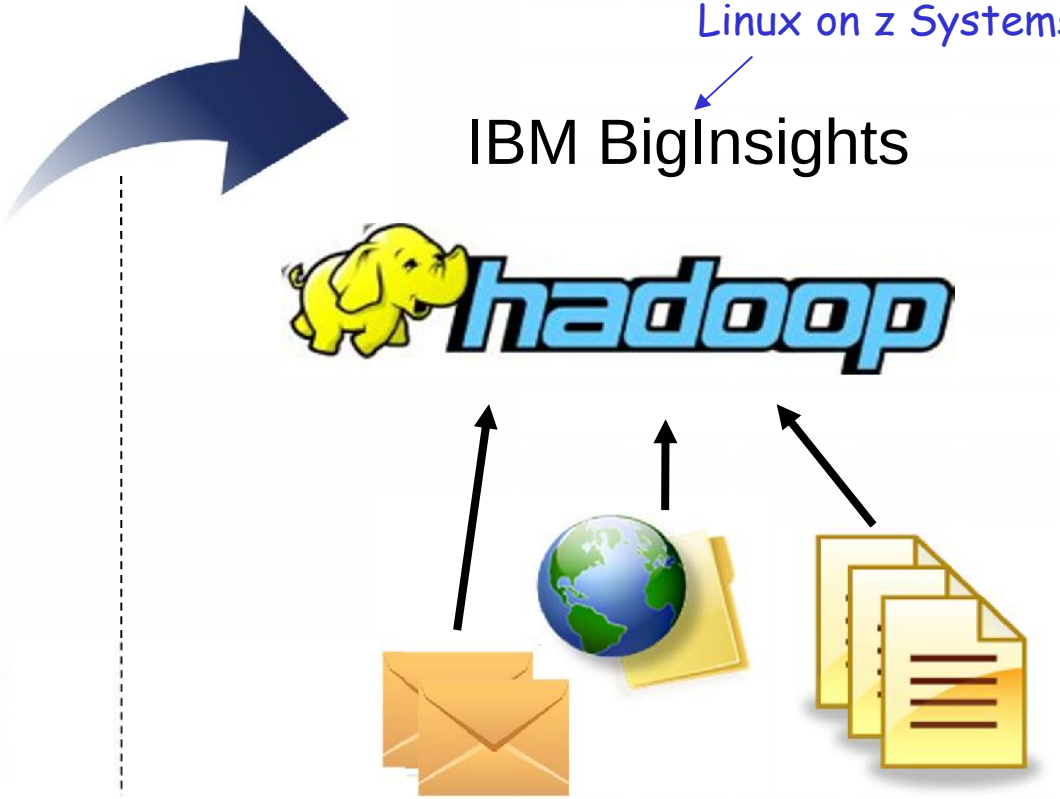
- Two main components
  - Hadoop distributed file system
  - MapReduce engine
    - Powerful, but tedious from a development perspective
    - “Like the assembly language of Hadoop”



DB2 11: new UDFs for Hadoop integration

Available now for Linux on z Systems

A new user-defined function (UDF) allows a data analytics job, specified in JAQL, to be submitted to a BigInsights server



A new table UDF reads the output of the analytics job and returns it in relational form

XQuery support for XML data

- Pre-DB2 11: XPath expressions can be used to navigate through XML documents and to address parts of XML documents
  - XPath is a subset of XQuery, which is a richer language for accessing XML documents
  - XPath limitations often necessitated using a mixture of XPath and SQL, and that could make query coding more difficult
- DB2 11 includes XQuery support, providing a richer set of XML expressions that can be used with the built-in functions XMLQUERY, XMLEXISTS, and XMLTABLE
  - Queries can be expressed purely using XQuery, versus a mixture of XPath and SQL, and that can boost programmer productivity
- XQuery support was retrofitted to DB2 10 via APARS PM47617 and PM47618

In conclusion…

- DB2 10 and 11 delivered a lot of new application-enabling features
  - How many of these are being used at your site?
  - How many *could* be put to good use at your site?

Thanks for your time!