IBM Software Group

DB2 9 SQL Enhancements

Fen-Ling Lin
Senior Technical Member Stuff and Manager
Query Technology, DB2 for z/OS
IBM Silicon Valley Laboratory

DB2 for z/OS 2009 Taipei Conference
OCT 5 - 6, 2009
Taipei, Taiwan

ON DEMAND BUSINESS

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, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS AND/OR SOFTWARE.

DB2 for z/OS V9 SQL, DB2 family & porting

- XML
- MERGE
- SELECT FROM UPDATE, DELETE, MERGE
- TRUNCATE
- INSTEAD OF TRIGGER
- BIGINT, VARBINARY, DECIMAL FLOAT
- Native SQL Procedure Language
- Optimistic locking
- LOB File reference variable & FETCH CONTINUE
- FETCH FIRST & ORDER BY in subselect and fullselect
- INTERSECT & EXCEPT
- ROLE & trusted context
- Many new built-in functions, caseless comparisons
- Index on expression
- Improved DDL consistency
- CURRENT SCHEMA

DB2 SQL

z z/OS V7
common
LUW Linux, Unix & Windows V8.2

z
- Range partitioning

common
- Inner and Outer Joins, Table Expressions, Subqueries, GROUP BY, Complex Correlation, Global Temporary Tables, CASE, 100+ Built-in Functions, Limited Fetch, Insensitive Scroll Cursors, UNION Everywhere, MIN/MAX Single Index Support, Self Referencing Updates with Subqueries, Sort Avoidance for ORDER BY, and Row Expressions, Call from trigger, statement isolation

LUW
- Updateable UNION in Views, ORDER BY/FETCH FIRST in subselects & table expressions, GROUPING SETS, ROLLUP, CUBE, INSTEAD OF TRIGGER, EXCEPT, INTERSECT, 16 Built-in Functions, MERGE, Native SQL Procedure Language, SET CURRENT ISOLATION, BIGINT data type, file reference variables, SELECT FROM UPDATE, DELETE & MERGE, multi-site join, 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

DB2 SQL

z z/OS V8
common
LUW Linux, Unix & Windows V8.2

z

Multi-row INSERT, FETCH & multi-row cursor UPDATE, Dynamic Scrollable Cursors, GET DIAGNOSTICS, Enhanced UNICODE for SQL, join across encoding schemes, IS NOT DISTINCT FROM, Session variables, range partitioning

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 Scroll Cursors, UNION Everywhere, MIN/MAX Single Index Support, Self Referencing Updates with Subqueries, Sort Avoidance for ORDER BY, and Row Expressions, 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, Call from trigger, statement isolation, FOR READ ONLY KEEP UPDATE LOCKS, SET CURRENT SCHEMA, Client special registers, long SQL object names, SELECT from INSERT

LUW

Updateable UNION in Views, ORDER BY/FETCH FIRST in subselects & table expressions, GROUPING SETS, ROLLUP, CUBE, INSTEAD OF TRIGGER, EXCEPT, INTERSECT, 16 Built-in Functions, MERGE, Native SQL Procedure Language, SET CURRENT ISOLATION, BIGINT data type, file reference variables, SELECT FROM UPDATE, DELETE & MERGE, multi-site join

DB2 SQL

z z/OS V9
common
LUW Linux, Unix & Windows V9

z

Multi-row INSERT, FETCH & multi-row cursor UPDATE, Dynamic Scrollable Cursors, GET DIAGNOSTICS, Enhanced UNICODE for SQL, join across encoding schemes, IS NOT DISTINCT FROM, Session variables, range partitioning, TRUNCATE, DECIMAL FLOAT, VARBINARY, optimistic locking, FETCH CONTINUE, ROLE, MERGE, SELECT from MERGE

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 Scroll Cursors, UNION Everywhere, MIN/MAX Single Index Support, Self Referencing Updates with Subqueries, Sort Avoidance for ORDER BY, and Row Expressions, 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, Call from trigger, statement isolation, FOR READ ONLY KEEP UPDATE LOCKS, SET CURRENT SCHEMA, Client special registers, long SQL object names, SELECT from INSERT, UPDATE, DELETE & MERGE, INSTEAD OF TRIGGER, Native SQL Procedure Language, BIGINT, file reference variables, XML, FETCH FIRST & ORDER BY in subselect and fullselect, caseless comparisons, INTERSECT, EXCEPT, not logged tables, range partitioning, compression

LUW

Updateable UNION in Views, GROUPING SETS, ROLLUP, CUBE, 16 Built-in Functions, SET CURRENT ISOLATION, multi-site join, MERGE, XQUERY

Key SQL Features in DB2 9

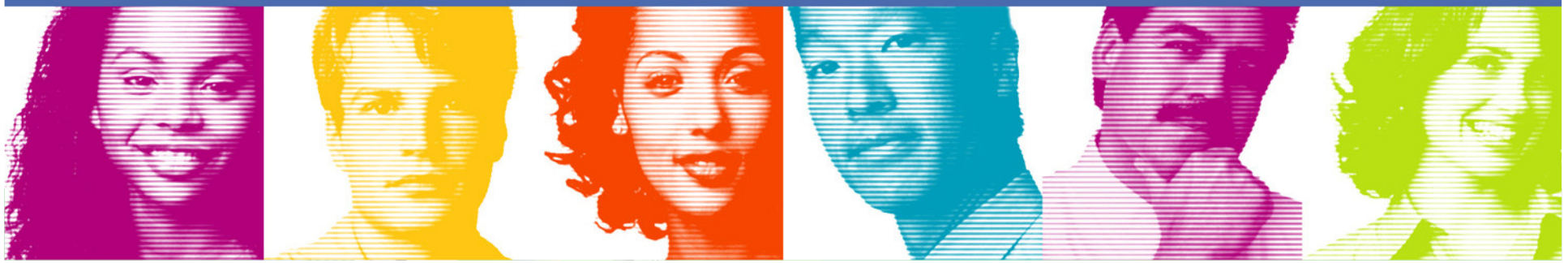
- Intersect/Except
- Instead of Trigger
- MERGE
- SELECT from MERGE, UPDATE, DELETE
- TRUNCATE
- ORDER BY and FETCH FIRST N Row in Subselect
- RANK, DENSE_RANK, ROW_NUMBER
- Index on Expression

Key SQL Features in V9

- FETCH Continue to aid fetching of LOB data
- LOB File Reference
- RENAME Column
- RENAME TABLE
- Automatic Creation of Objects
- New Data Types
- XML

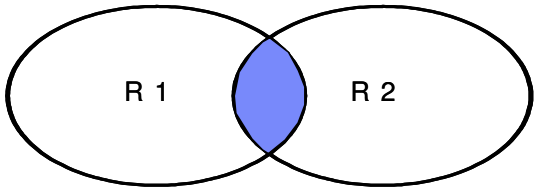
Intersect and Except

Act.Right.Now.

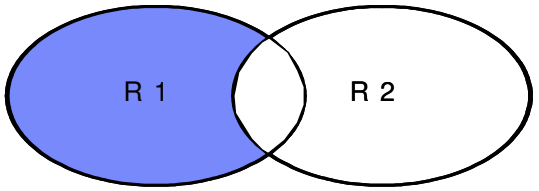


INTERSECT/EXCEPT

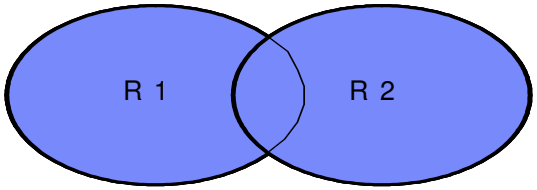
- SET operator: UNION, **INTERSECT, EXCEPT**



INTERSECT



EXCEPT
(Difference)



UNION

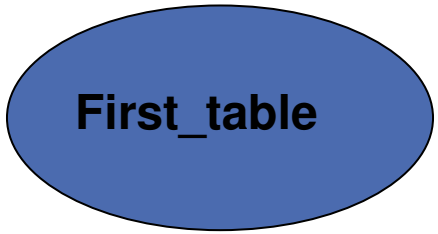
```
SELECT c11, c12, ?  c1n FROM T1 <set-op>
SELECT c21, c22, ?  c2m FROM T2
```

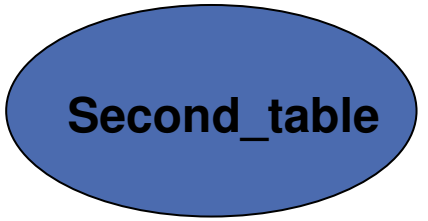
Columns participating in INTERSECT and EXCEPT

- R1 and R2 must have the same number of columns
  - Data type for the n-th column of R1 must be compatible with the n-th column of R2
  - Data type must no be CLOB, BLOB, DBCLOB, XML, or distinct type based on these type
- Qualified column names cannot be used in the ORDER BY clause with the set operators are specified

Sample query

```
SELECT LAST_NAME,
FIRST_NAME, …
        FROM  first_table
        WHERE ….

UNION  |  INTERSECT  |  EXCEPT

SELECT LAST_NAME,
FIRST_NAME, …
        FROM second_table
        WHERE ….
```





Result of Operations -- R1 UNION R2

R1	R2	UNION ALL	UNION
1	1	1	1
1	1	1	2
1	3	1	3
2	3	1	4
2	3	1	5
2	3	2	
3	4	2	
4		2	
4		3	
5		3	
		3	
		3	
		3	
		4	
		4	
		4	
		5	

Show me all the rows from The result table of each SELECT statement

Result of Operations -- R1 EXCEPT R2

R1	R2	EXCEPT ALL	EXCEPT
1	1	1	2
1	1	2	5
1	3	2	
2	3	2	
2	3	4	
2	3	5	
3	4		
4			
4			
5			

Show me all the rows in R1
Which do not have a corresponding
Row in R2

Result of Operations -- R1 INTERSECT R2

R1	R2	INTERSECT ALL	INTERSECT
1	1	1	1
1	1	1	3
1	3	3	4
2	3	4	
2	3		
2	3		
3	4		
4			
4			
5			

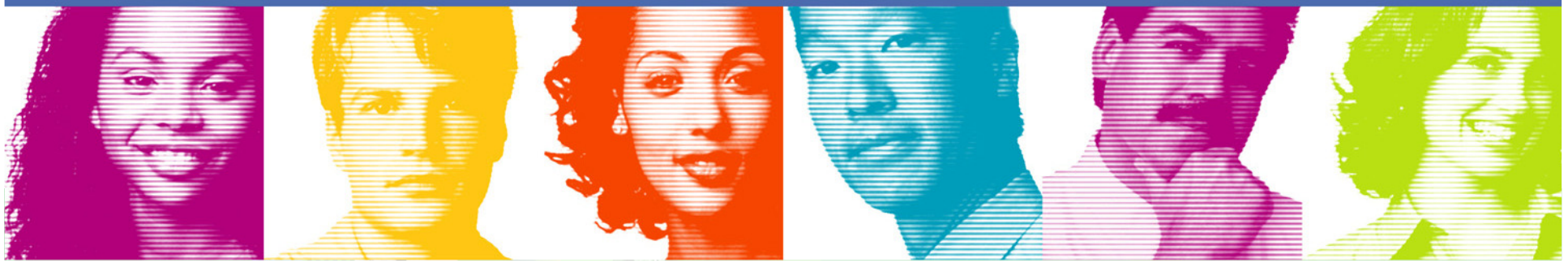
Show me all the rows which Appear in both R1 and R2

INTERSECT/EXCEPT

R1	R2	UNION ALL	UNION	EXCEPT ALL	EXCEPT	INTERSECT ALL	INTERSECT
1	1	1	1	1	2	1	1
1	1	1	2	2	5	1	3
1	3	1	3	2		3	4
2	3	1	4	2		4	
2	3	1	5	4			
2	3	2		5			
3	4	2					
4		2					
4		3					
5		3					
		3					
		3					
		3					
		4					
		4					
		4					
		5					

INSTEAD OF TRIGGERS

Act.Right.Now.



INSTEAD OF Triggers: current problem and goal

- Customers use views for read access control
- Many views are not updatable, so customers have to access base tables for data changes.
- No INSERT / UPDATE / DELETE for read-only views
- **Goal:** to provide a mechanism to unify the target for all read / write access by an application (i.e., through views)

Instead of Trigger

- A new type of trigger (~ BEFORE, AFTER triggers)
- Defined on VIEWs
  - provides an extension to the updatability of views
  - requested update operation against the view gets replaced by the trigger logic
  - application still believes all operations are performed against the view
  - applicable even for updatable views

Instead of Trigger

```
CREATE TABLE WEATHER (CITY VARCHAR(25), TEMPF DECIMAL(5,2));
CREATE VIEW CELCIUS_WEATHER_V (CITY, TEMPC) AS
      SELECT CITY, (TEMPF-32)*5.00/9.00 FROM WEATHER

CREATE TRIGGER CW_INSERT INSTEAD OF INSERT ON
       CELCIUS_WEATHER_V
REFERENCING NEW AS NEWCW DEFAULTS NULL
FOR EACH ROW MODE DB2SQL
   INSERT INTO WEATHER VALUES (NEWCW.CITY,
                               9.00/5.00*NEWCW.TEMPC+32)

CREATE TRIGGER CW_UPDATE INSTEAD OF UPDATE ON
       CELCIUS_WEATHER_V
REFERENCING NEW AS NEWCW OLD AS OLDCW DEFAULTS NULL
FOR EACH ROW MODE DB2SQL
   UPDATE WEATHER AS W
     SET W.CITY = NEWCW.CITY,
         W.TEMPF = 9.00/5.00*NEWCW.TEMPC+32
     WHERE W.CITY = OLDCW.CITY
```

DROP TRIGGER / VIEW

- DROP view also drops INSTEAD OF triggers
- DROP trigger invalidates other packages (including trigger packages) that depends on the dropped INSTEAD OF trigger

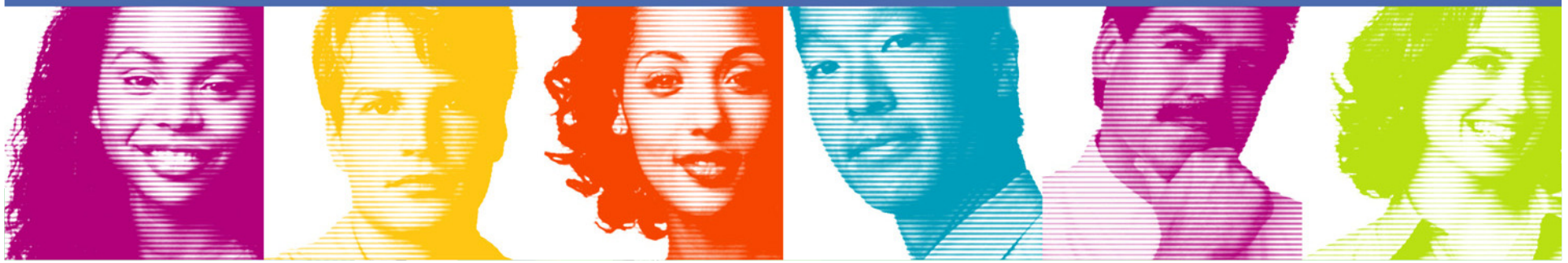
```
Create trigger TR1 instead of update on V1
    begin … end

Create trigger TR2 after update on T1
    begin
        Update v1 …              -> TR2 depends on TR1
    end

DROP  trigger TR1     -→ package TR2 is invalidated
```

Restrictions

- Only 1 INSTEAD OF INSERT, UPDATE, DELETE per view
- View cannot be symmetric (i.e,, no WHERE clause)
- Only has row granularity
- No WHEN clause
- Cannot specify UPDATE OF column list
- New REFERENCING DEFAULTS NULL clause
- Cannot change transition variables
- Does not work with position UPDATE/DELETE
- No LOB, XML
- SELECT FROM UPDATE/DELETE/INSERT not supported
- MERGE into a view with INSTEAD OF trigger is not supported

MERGE

Act.Right.Now.



Choices in V8

- Issue a SELECT to determine whether the row exists
  - If Yes, UPDATE
  - If No, INSERT
- Determine whether or not the row is likely to exist most of the time
  - If the row is likely to exist, try UPDATE – if it fails, perform INSERT
  - If the row is not likely to exist, try INSERT – if it fails, perform UPDATE
- In either case, more than one SQL statement is necessary to perform the data changes

MERGE

- Combine UPDATE and INSERT operation to a target table or view, from a input source of host-variable-arrays modeled as a source table
  - When source rows match the target, UPDATE the target rows from source
  - When source rows do not match to target, INSERT source rows into target
  - UPDATE/INSERT triggers will be fired

Example

Source

S.id	S.amt
1	30
5	10
10	40
5	20
1	50

Target

Account - changed

T.id	balance
1	1030
5	10
10	540
5	30
1	1080

Account - before

T.id	balance
1	1000
10	500
200	600
300	300
315	100
500	4000
...	

```
MERGE  INTO account AS T
USING VALUES (:hv_id, :hv_amt) FOR 5 ROWS  AS S(id,amt)
ON T.id = S.id
WHEN MATCHED THEN
   UPDATE SET balance = T.balance + S.amt
WHEN NOT MATCHED THEN
   INSERT (id, balance) VALUES (S.id, S.amt)
NOT ATOMIC CONTINUE ON SQLEXCEPTION
```

Account - after

T.id	balance
1	1080
5	30
10	540
200	600
300	300
315	100
500	4000
...	

EXPLAIN changes

- Plan_table
  - New QBLOCK_TYPE: “MERGE”
  - MERGE is QB(1)
  - UPDATE is QB(2)
  - INSERT is QB(3)

- DSN_STATEMENT_TABLE
  - New STMT_TYPE of “MERGE”

Sample Explain

qblockno	qblock_type	planno	correlation _name	table_type	join_type	method	accesstype
1	MERGE	1	S	B (1*)			V (2*)
1	MERGE	2	T	T	L	1 (3*)	(4*)
2	UPDATE	1	T	T			
3	INSERT	1	T	T			

- 1* : table_type of "B" is already supported in V8
  - Need to update the "EXPLAIN" statement description in SQL Reference
- 2* : accesstype of "V" is already supported in V8
  - Need to update the "EXPLAIN" statement description in SQL Reference
- 3* : Since we are doing "update in place",
  - only Nested Loop Join is considered
- 4* : Since we are doing "update in place",
  - if an index column is being updated, the index won't be considered for the table access to avoid Halloween problem
  - RID access ("I" with prefetch="L") won't be considered
  - Sparse index access ("T") won't be considered
- No parallel support for MERGE.

Merge notes

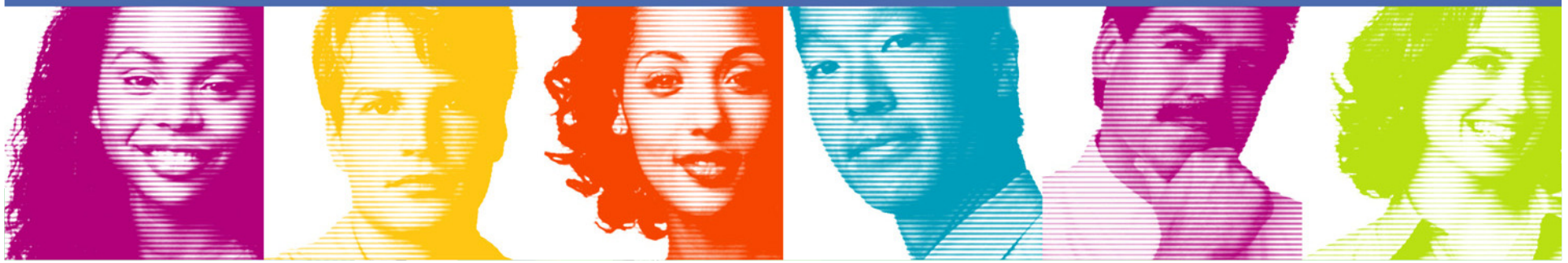
- Source data are piped into target
  - A row inserted into target is immediately available for update
  - A row updated is immediately available for more update in the same statement
- NOT atomic – operation continues to next input, even after the merge operation of an input row fails

GET DIAGNOSTICS is useful for operations!!!

- No MERGE trigger; UPDATE/INSERT trigger will be fired
- If target is a view with INSTEAD OF triggers, MERGE is not allowed

SELECT FROM MERGE, UPDATE, DELETE

Act.Right.Now.



Review: V8 – SELECT FROM INSERT

- Benefits
  - Enhances usability and power of SQL
  - Enhances user to immediately determine values inserted in tables by DB2 (identify, 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 column, sequence values
    - User-defined defaults, expressions
    - Column modified by BEFORE INSERT triggers
    - ROWIDs

Example of SELECT FROM INSERT

```
DECLARE CS1 CURSOR FOR
   SELECT  EMP_ROWID
      FROM FINAL TABLE
  (INSERT INTO DSN810.EMP_RESUME(EMPNO)
      SELECT EMPNO FROM DSN810.EMP));
```

SELECT FROM UPDATE/DELETE/MERGE

- SELECT from UPDATE or DELETE will be implemented by allowing a searched UPDATE or searched DELETE statement in the FROM clause of a select-statement that is a subselect or in the SELECT INTO statement. By allowing a searched UPDATE or searched DELETE to appear in a select-statement or SELECT INTO statement, the database will allow the user to know which values were updated in a table and which rows were deleted from a table via a single SQL statement.

- SELECT FROM MERGE will return all the updated rows and inserted rows, including column values which are generated by DB2.

- An INCLUDE column specified is being introduced to allow the user to identify a new column for the select-list and as a method for sorting the data (also added to SELECT from INSERT).

SELECT FROM MERGE/UPDATE/DELETE

- V8 - The INSERT statement was allowed in the FROM clause
- V9 - A searched UPDATE/DELETE is now allowed in the FROM clause

Delete employees at level 'Contractor' and return the total amount of salary:

```
SELECT SUM(Salary)
FROM OLD TABLE
(DELETE FROM Employee WHERE Level =
'Contractor');
```

SELECT FROM MERGE/UPDATE/DELETE

Update salaries of employees at level 'Associate' and return the new salary:

```
SELECT Name, Salary
FROM FINAL TABLE
(UPDATE Employee   SET Salary = Salary *1.1
  WHERE Level = 'Associate');
```

Update salaries of employees at level 'Associate' and return the old salary:

```
SELECT Name, Salary
FROM OLD TABLE
(UPDATE Employee   SET Salary = Salary *1.1
  WHERE Level = 'Associate');
```

INCLUDE Columns

- Introduces a list of columns to be included in the result table of the DELETE/INSERT/UPDATE/MERGE statement.

- The include columns are only available if the DELETE / INSERT / UPDATE / MERGE statement is nested in the from clause of a select-statement or SELECT INTO statement.

Example - select from Final Table (Merge...)

Source

S.id	S.amt
1	30
5	10
10	40
5	20
1	50
99	90

Account - changed (plan C)

T.id	balance	status
1	1030	upd
5	10	ins
10	540	upd
5	30	upd
1	1080	upd
99	90	ins

Account - target table

T.id	balance
1	1000
10	500
200	600
300	300
315	100
500	4000
...	

```
SELECT balance, status  FROM FINAL TABLE (
MERGE INTO account AS T   INCLUDE( status char(3))
USING VALUES (:hv_id, :hv_amt) FOR 3 ROWS  AS S(id,amt)
ON T.id = S.id
WHEN MATCHED THEN
   UPDATE SET balance = T.balance + S.sum_amt,
                status    = 'upd'
WHEN NOT MATCHED THEN
   INSERT (id, balance) VALUES (S.id, S.sum_amt,'ins')
NOT ATOMIC CONTINUE ON SQLEXCEPTION
)
```

include columns

output (plan B)

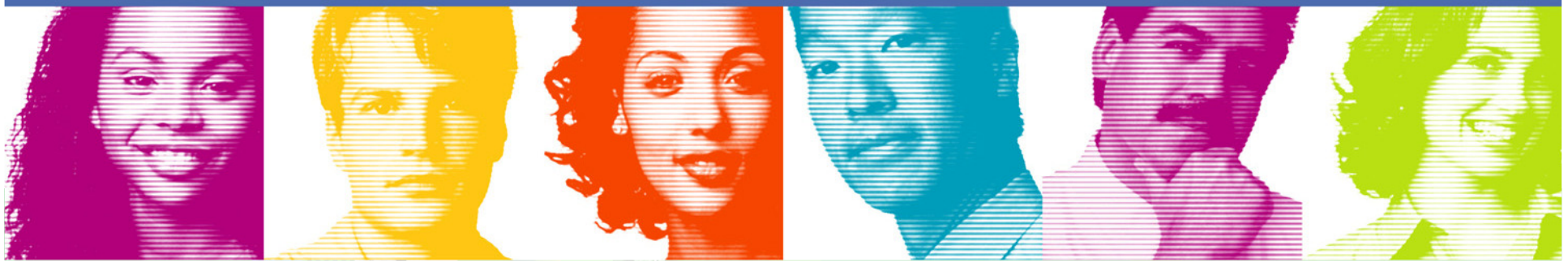
T.id	balance	status
1	1080	upd
5	30	upd
10	540	upd
99	90	ins

Account - after

T.id	balance
1	1080
5	30
10	540
99	90
200	600
300	300
315	100
500	4000
...	

ORDER BY and FETCH FIRST N Rows in Subselect

Act.Right.Now.



Background

- Prior to V9, DB2 z/OS prohibit ORDER BY and FETCH FIRST n Rows in a select
  - i.e., one can write

```
SELECT * FROM T ORDER BY c1 FETCH FIRST 1 ROW ONLY;
```

  - But cannot write


```
INSERT INTO TEMP
    SELECT * FROM T ORDER BY c1 FETCH FIRST 1 ROW ONLY;
```

In V9

- Allow all semantically relevant clause of the select statement to be pushed into subqueries. The original query can be taken as is and wrapped by more SQL, such as show in the example above
- Provides more function by being able to select, e.g., the top N rows in a leg of a join, a leg of union, or a subquery

```
(SELECT * FROM T1 ORDER BY C1
      FIRST 4 ROW ONLY)
  UNION
SELECT * FROM T2 ;
```

Customer Requirement

- One customer has a huge table of which they want just the first 2000 rows sorted in a particular order.
- If the sort is done first, and the FETCH FIRST later, it will cause a huge sort for no reason.
- The solution is the V9 FETCH FIRST N Row in Subselect
-

```
SELECT A, B, C FROM
  (SELECT A, B, C, FROM TABLE_A  WHERE ….
      FETCH FIRST 2000 ROWS ONLY) AS TABLE_B
  ORDER BY C, B;
```

ORDER BY and FETCH FIRST in subselect

- ORDER BY clause can be specified in subselect or fullselect
- FETCH FIRST n ROWS ONLY clause can be specified in subselect or fullselect
- ORDER OF table-designator extension to the ORDER BY clause

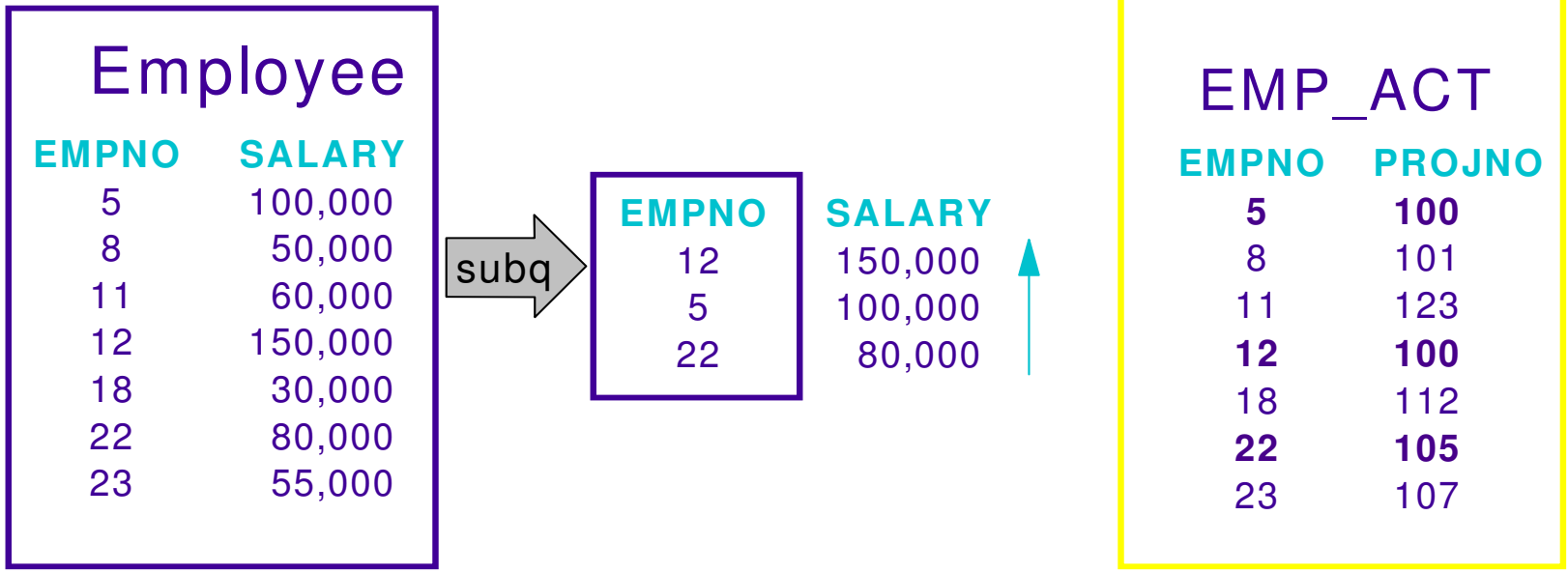
```
(SELECT * FROM T1
 ORDER BY C1)
UNION
(SELECT * FROM T2
 ORDER BY C2
 FETCH FIRST 2 ROWS)
```

```
(SELECT * FROM T1
 ORDER BY C1)
UNION
SELECT * FROM T2
ORDER BY C2
FETCH FIRST 2 ROWS
```

Example

Using the EMP_ACT table, find the project numbers that have an employee whose salary is in the top 3 of all employees.

```
SELECT EMP_ACT.EMPNO, PROJNO
FROM EMP_ACT
WHERE EMP_ACT.EMPNO IN
   (SELECT EMPLOYEE.EMPNO
     FROM EMPLOYEE
     ORDER BY SALARY DESC
     FETCH FIRST 3 ROWS ONLY )
```



ORDER OF table-designator

The use of ORDER OF table-designator in the ORDER BY clause in a nested table expression allows the higher level select to retain the ordering of the rows returned from the nested table expression

```
SELECT C1 FROM
  (SELECT C1, C2 FROM T1
   UNION ALL
   SELECT C1, C2 FROM T2
   ORDER BY C1) AS UTABLE
ORDER BY ORDER OF UTABLE
```

The higher select "inherits" the ordering of the rows of the result table of the inner table expression (**UTABLE**).

```
SELECT TEMP.Cx, TEMP.Cy, T1.C1, T1.C2
FROM T1 ,
        (SELECT T2.C1, T2.C2
         FROM T2
         ORDER BY 2) AS TEMP(Cx,Cy)
WHERE Cy = T1.C1
ORDER BY ORDER OF TEMP
```

```
SELECT TEMP.Cx, TEMP.Cy, T1.C1, T1.C2
FROM T1 ,
        (SELECT T2.C1, T2.C2
         FROM T2
         ORDER BY 2) AS TEMP(Cx,Cy)
WHERE Cy = T1.C1
ORDER BY TEMP.Cy
```

Examples

```
(SELECT * FROM T1
 ORDER BY C1)
UNION
(SELECT * FROM T2
 ORDER BY C2
 FETCH FIRST 2 ROWS)
```

```
(SELECT * FROM T1
 ORDER BY C1)
UNION
SELECT * FROM T2
ORDER BY C2
FETCH FIRST 2 ROWS
```

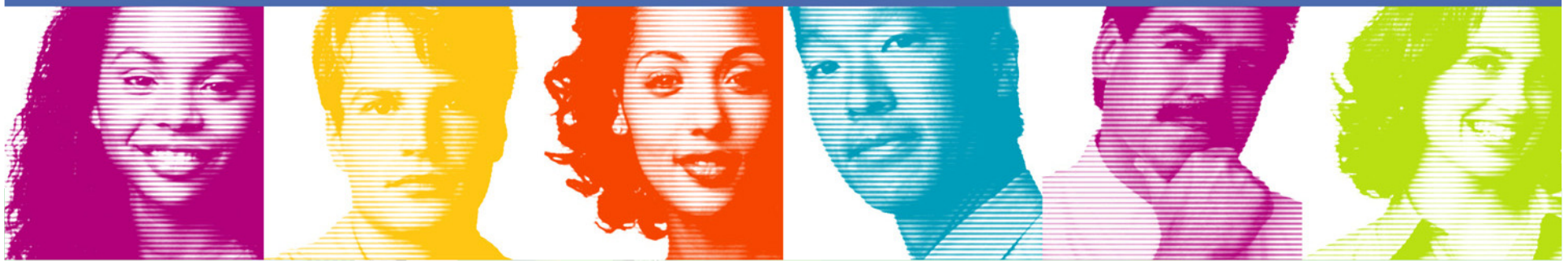
The following examples are **invalid**

```
SELECT * FROM T1
ORDER BY C1
UNION
(SELECT * FROM T2
 ORDER BY C2
 FETCH FIRST 2 ROWS)
```

```
CREATE VIEW V1 AS
(SELECT * FROM T1
ORDER BY C1)
```

TRUNCATE

Act.Right.Now.



What TRUNCATE does

- Gives user an alterative way of emptying a table, with more flexibility over the current DELETE statement with no WHERE clause (i.e., a mass delete operation):
  - Delete all data rows in a designated DB2 table without activating DELETE trigger
  - DB2 catalog definition of the table (i.e., dropping and recreating of the delete triggers) is not needed for faster processing
  - Provides an option to allow the users to empty the designated DB2 table permanently without going through the current commit phase
  - Provides an option to reuse deallocated storage

TRUNCATE Table

- A fast way to empty a table
- **DELETE Triggers are ignored**
- Indexes, LOB, XML Tablespaces are also deleted
- X lock on the target table, Mass-delete

```
TRUNCATE <TABLE> TABLE-NAME

  < DROP STORAGE | REUSE STORAGE>

  < RESTRICT WHEN DELETE TRIGGERS | IGNORE DELETE TRIGGERS>

  < IMMEDIATE>
```

Processing modes for TRUNCATE

- Normal way:
  - Truncate operation process each data page to physically delete data records from the page
    - Table in a simple table space
    - Table in a partitioned table space
    - Any table with table attribues
      - CDC-enabled (Change Data Capture)
      - MLS-enabled (Multiple Level Security)
      - VALID PROC exist
- Fast way:
  - Truncate operation deletes data records without physically processing each data page
    - table in a segmented table without table attributes
    - table in a universal table space without table attributes

Examples

- Empty an old inventory table regardless any existing DETETE triggers and like to return its allocated space.

```
TRUNCATE INVENTORY_TABLE
    IGNORE DELETE TRIGGERS
    DROP STORAGE;
```

- Empty an old inventory table regardless any existing DELETE triggers but also like to preserve its allocated space for later user

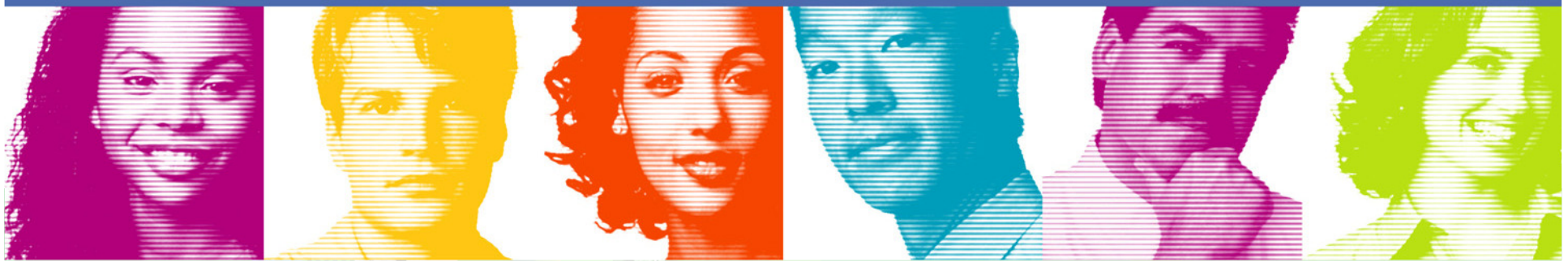
```
TRUNCATE INVENTORY_TABLE
    IGNORE DELETE TRIGGERS
    REUSE STORAGE;
```

TRUNCATE …. IMMEDIATE

- Specifies that the truncate operation is processed immediately and cannot be undone
- When IMMEDIATE option is specified, the table must not contain any uncommitted updates:
  - For a DGTT table object, the IMMEDIATE option does not apply to it. The truncate operation will fail since the table space contains a DGTT will be always in the update mode.
  - No uncommitted DDL is allowed on the table prior to the TRUNCATE
- The truncated table is immediately available for use in the same unit of work
- Although a ROLLBACK statement is allowed after the TRUNCATE statement, the truncate operation is not undone, and the table rename truncated. Other data changes following TRUNCATE are rolled back

Rank, DESE_RANK, Row_Number

Act.Right.Now.



OLAP specification -- RANK, DENSE_RANK, ROW_NUMBER

- RANK() OVER Window ----> OLAP Function
  - PARTITION BY sh.territory --- row should be assigned to partition according to territory
  - ORDER BY sh.sales --- row sorted in the order of sales amount within each partition
- Apply after Join, Predicates, Group By, Having
- A new class of aggregate functions
  - Rank
  - DENSERANK
  - ROWNUMBER

```
SELECT  sh.territory, sh.sales,
        Rank() over (PARTITION BY sh.territory
                     ORDER BY sh.sales desc) as rank
   FROM sales_history;
```

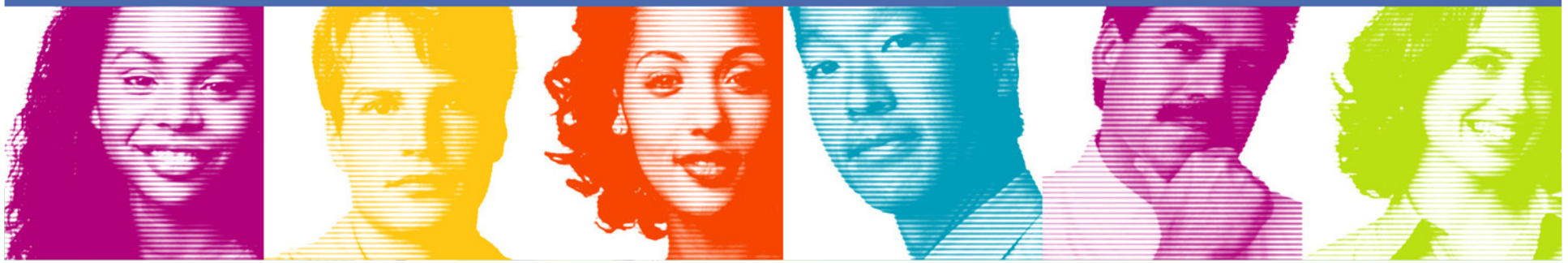
OLAP specification -- RANK, DENSE_RANK, ROW_NUMBER

```
SELECT EMPNUM, DEPT, SALARY,
    RANK()       OVER (ORDER BY SALARY DESC) as RANK,
    DENSERANK()  OVER (ORDER BY SALARY DESC)as DENSERANK,
    ROWNUMBER()  OVER (ORDER BY SALARY DESC) as ROWNUM
FROM EMPLOYEE;
```

EMPNUM	DEPT	SALARY	RANK	DENSERANK	ROWNUM
3	-	84000	1	1	1
8	3	79000	2	2	2
6	1	78000	3	3	3
2	1	75000	4	4	4
7	1	75000	4	4	5
12	3	75000	4	4	6
10	3	55000	7	5	7
11	1	53000	8	6	8

New Data Type

Act.Right.Now.



New data types: BIGINT, BINARY, VARBINARY, DECFLOAT

- **BIGINT** - big integer.
  - Big integer is a binary integer with a precision of 63 bits. The range of big integers is [-9223372036854775808, 9223372036854775807
- **BINARY** ? fixed-length binary string.
  - Fixed-length binary string is in a range of [1,255]. The padding with hexadecimal zeros (X? 0?. Not associated with any CCSID
- **VARBINARY** ? varying-length binary string.
  - Varying-length binary string is in a range of [1,32704]. No padding is performed. Not associated with any CCSID
- **DECFLOAT** ? Decimal float.
  - DECFLOAT(16) = decimal64 format (8 bytes)
  - DECFLOAT(34) = decimal128 format (16 bytes)

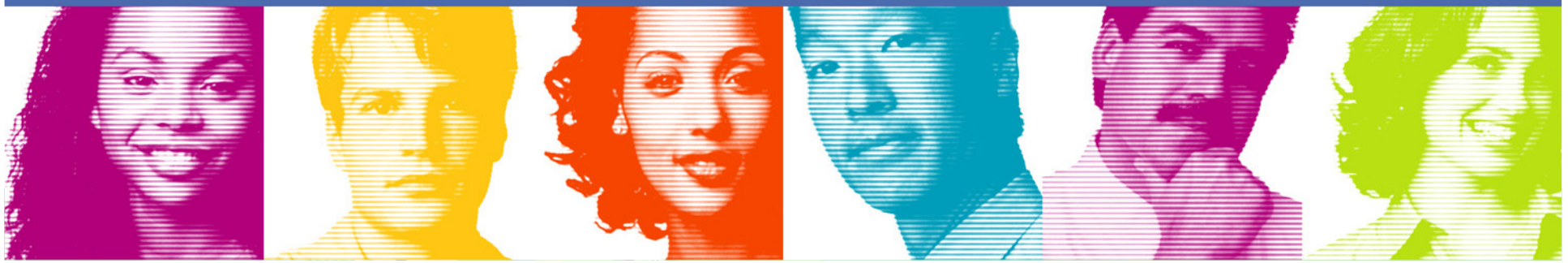
DECFLOAT

- Both IEEE and hexadecimal floating point numbers can only approximate common decimal numbers. But DFP can represent decimal number exactly.
- DFP can represent much bigger and smaller number than DECIMAL.

Description	Limit
Smallest DECFLOAT(16) Values	$-9.999999999999999 \times 10^{384}$
Largest DECFLOAT(16) Value	$9.999999999999999 \times 10^{384}$
Smallest positive DECFLOAT(16) Value	$1.000000000000000 \times 10^{-383}$
Largest negative DECFLOAT(16) value	$-1.000000000000000 \times 10^{-383}$
Smallest DECFLOAT(34) Value	$-9.999999999999999999999999999999999 \times 10^{6144}$
Largest DECFLOAT(34) Value	$9.999999999999999999999999999999999 \times 10^{6144}$
Smallest positive DECFLOAT(34) Value	$1.000000000000000000000000000000000 \times 10^{-6143}$
Largest negative DECFLOAT(34) Value	$-1.000000000000000000000000000000000 \times 10^{-6143}$

Index on Expression

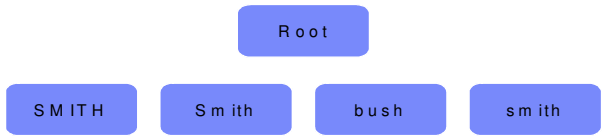
Act.Right.Now.

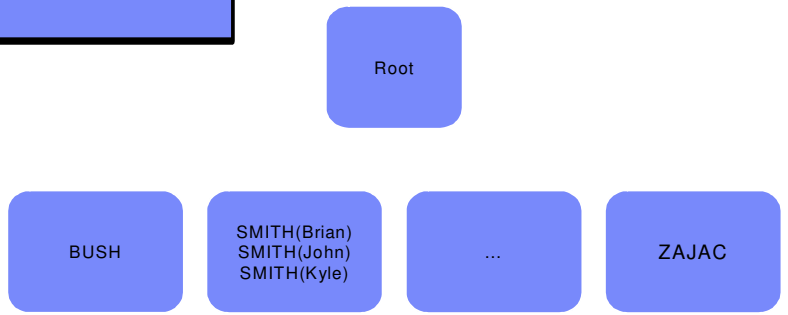


Index on Expression

- Create index on result of Expression
- Enhance Query Performance
- If we want to search for customers whose Upper(Lastname) = MITH□

```
CREATE INDEX IX_LastName ON CUSTOMER
 ( UPPER (Lastname), CUSTOMER_ID);

SELECT * FROM EMP
    WHERE UPPER (Lastname) = 'SMITH';
```

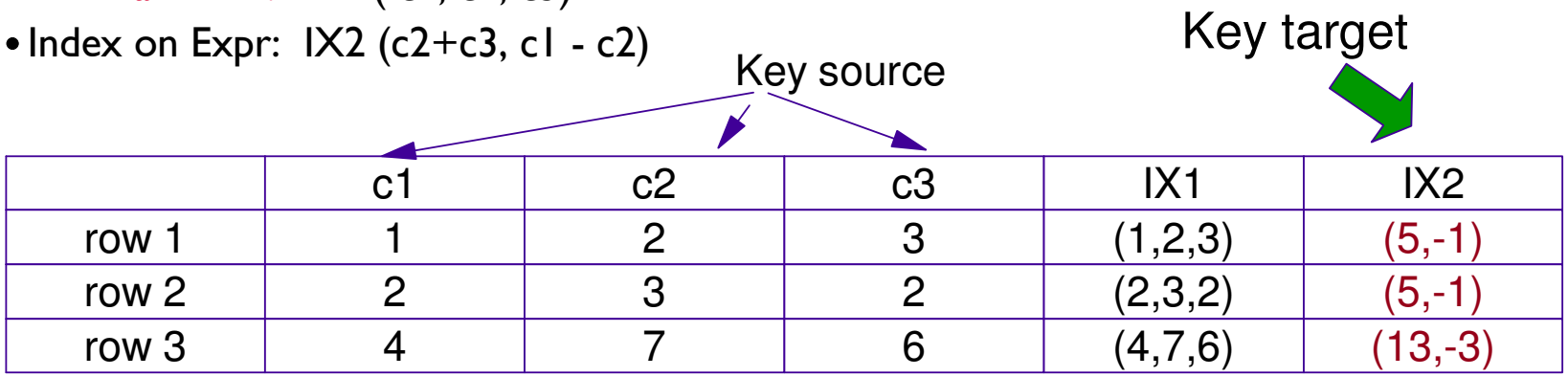




High Level Design(key target)

•Key source Key target

- Normal index: IX1(c1, c2, c3)
- Index on Expr: IX2 (c2+c3, c1 - c2)



	c1	c2	c3	IX1	IX2
row 1	1	2	3	(1,2,3)	(5,-1)
row 2	2	3	2	(2,3,2)	(5,-1)
row 3	4	7	6	(4,7,6)	(13,-3)

•Extended index

Unique

- Uniqueness — result of the expressions

  - In our example, IX2 is not an unque index although IX1 is.

    - ✓ Normal index: IX1(c1, c2, c3)
    - × Index on Expr: IX2 (c2+c3, c1 - c2)

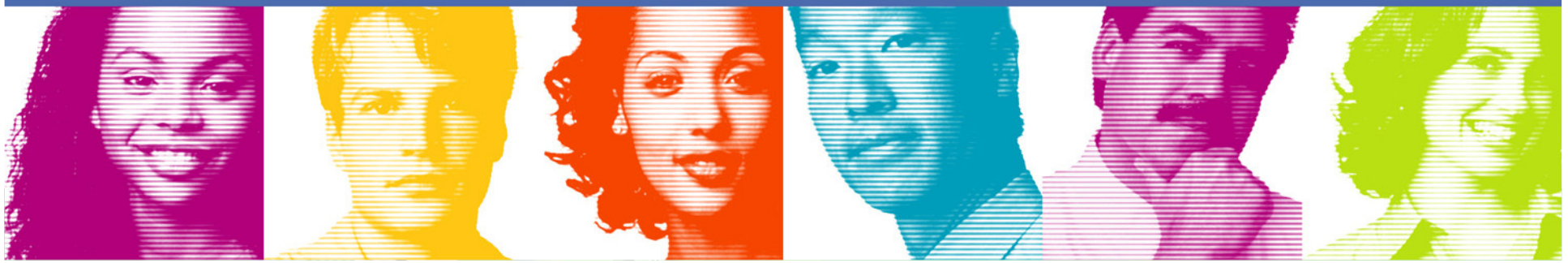
	c1	c2	c3	IX1	IX2
row 1	1	2	3	(1,2,3)	(5,-1)
row 2	2	3	2	(2,3,2)	(5,-1)
row 3	4	7	6	(4,7,6)	(13,-3)

Index on Expression - 2

- CREATE INDEX IX1 ON T1 (HEX(c1), BINARY(LTRIM(c2)));
- CREATE INDEX IX2 ON T2 (SUBSTR(c2, 1, 23), CONCAT(c2, c3));
- CREATE INDEX IX3 ON T2 (salary, bonus/salary, bonus+salary);
- CREATE INDEX IX4 ON T1 (DAYOFYEAR(endship) - DAYOFYEAR(startship));
- CREATE INDEX IX5 ON T2 (GRAPHIC(c3));
- CREATE INDEX IX6 ON T1 (VARCHAR(INSERT(vchar30,1,0,''),20));
- CREATE INDEX IX7 ON T1 (posstr(lvcharx2, '7.2E+02')) ;
- CREATE INDEX IX8 ON T1 (MIDNIGHT_SECONDS(birthday));

LOB File Reference Variable

Act.Right.Now.



Customer Pain Point

- Difficult to Load/Unload large Lob
- Poor Performance
- Significant application storage required
- Support File Reference Variable exists in other platforms

What is File Reference Variable

- A variable defined in the host language. It contains a file name and allows direct transfer of LOB data between DB2 and the file.
- Language Support
  - C, C++, JAVA
  - Cobol, PL/I
  - Assembler, REXX

Technical Overview

- Allow a Large LOB or XML to be read and write directly from a file
- Application no long needs to allocate storage to contain LOB or XML data
- Bypass the host language limitations on the maximum allowed size for LOB in the working storage
- Support HFS or BSAM
- Application must ensure DB2 has access to the file
- Three new SQL host variables
  - BLOB_FILE
  - CLOB_FILE
  - DBCLOB_FILE
- XML File Reference Variable
  - Specify SQL Type as XML AS

How does File Reference Variable work?

- Application declare a file reference variable

```
EXEC SQL BEGIN DECLARE SECTION
  SQL TYPE IS CLOB_FILE  hv_text_file;
EXEC SQL END DECLARE SECTION
```

- Precompiler building a host language construct as following:

```
struct {
            unsigned long name_length   // file name length
            unsigned long data_length    // data length
            unsigned long file_options    //  file options
            char              name[255]     // file name
      } hv_text_file;
```

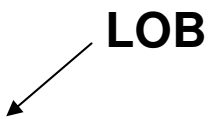
Input File Reference Variable

```
strcpy(hv_text_file.name, “/u/gainer/papers/sigmod.94");
hv_text_file.name_length = strlen("/u/gainer/papers/sigmod.94");
hv_text_file.file_options = SQL_FILE_READ;
```


```
EXEC SQL INSERT INTO PATENTS(TITLE,TEXT)
        VALUES(:hv_patent_title, :hv_text_file);
```

Output File Reference Variable

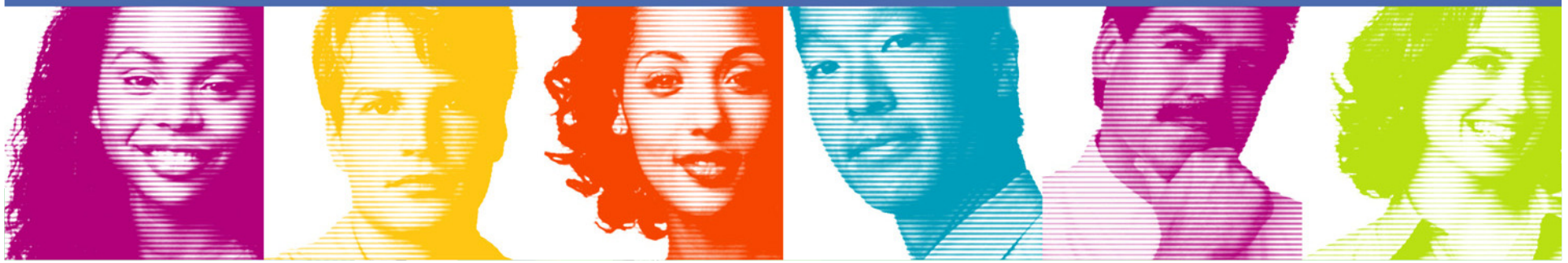
```
strcpy(hv_text_file.name, "/u/gainer/papers/sigmod.94");
hv_text_file.name_length = strlen("/u/gainer/papers/sigmod.94");
hv_text_file.file_options = SQL_FILE_CREATE;
```




```
EXEC SQL SELECT content INTO :hv_text_file from
    papers_table
 where TITLE = 'The Relational Theory behind Juggling';
```

Fetch Continue to Aid Fetching of Lob

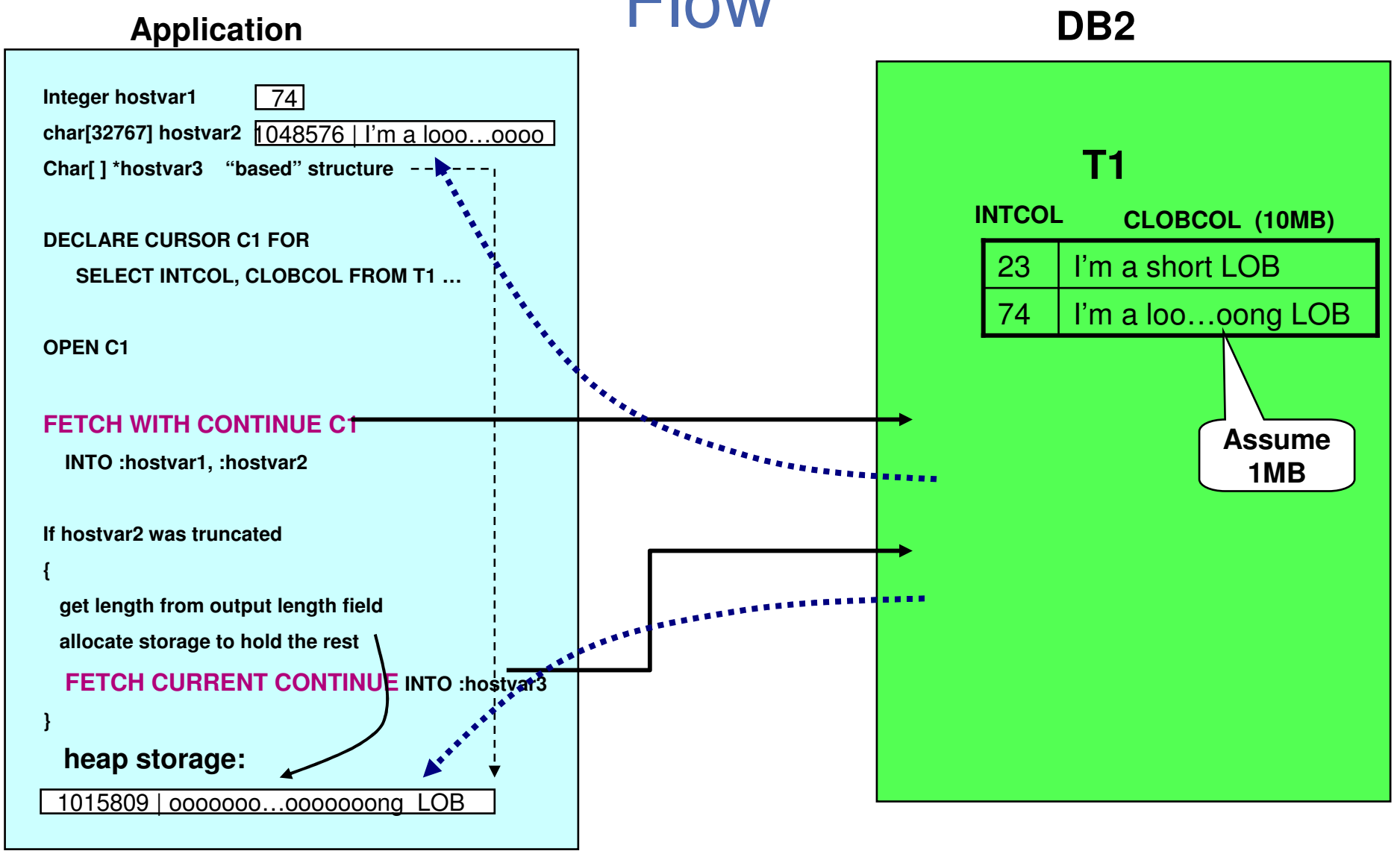
Act.Right.Now.



FETCH CONTINUE

- Provides 2 new syntax extensions on FETCH for LOB and XML data:
  - **FETCH WITH CONTINUE**
    - Just like a regular FETCH but
    - Tells DB2 how to react when truncation occurs on output of a LOB or XML column
    - Preserve the rest of the data, remember positiion
    - In the output length field, return the size that the host variable should have been
    - No effect on VARCHAR column truncation
  - **FETCH CURRENT CONTINUE**
    - Tells DB2 to continue fetching from the truncation point
    - The CURRENT keyword implies tay on the same row
    - Can be executed multiple times to tream?the data
    - ow-based?operation ? DB2 tries retrieve all truncated LOB/XML columns.
    - Column-based operation can be achieved by setting output lengths to zero for other columns

FETCH CONTINUE – Basic Application Flow

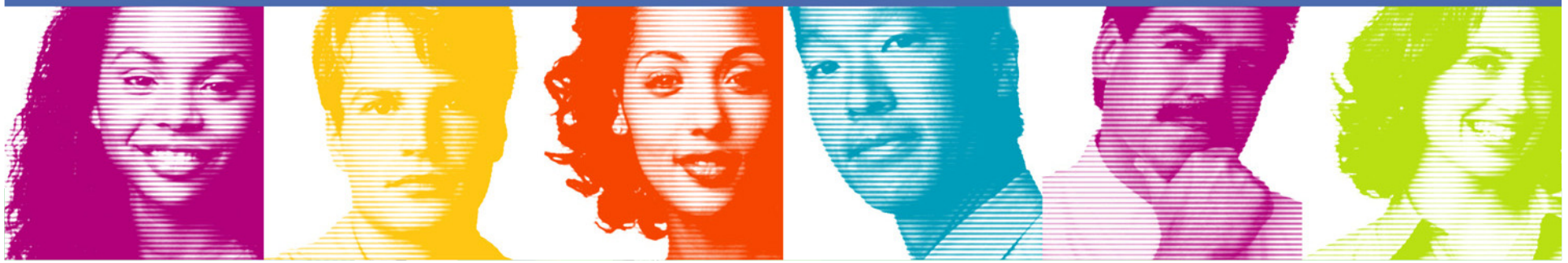


FETCH CONTINUE for XML and LOB

- No size associated with XML values
- Hard to allocate large memory
- Shortcomings with LOB Locator
- New FETCH CONTINUE statements: (one of two ways)
  - DECLARE CURSOR1 CURSOR FOR SELECT C2 FROM T1;
  - OPEN CURSOR1;
  - FETCH WITH CONTINUE CURSOR1 into :clobhv;
  - if (sqlcode >= 0) & sqlcode <> 100
  - Loop if truncation occurs until lob/xml complete (total length)
  - FETCH CURRENT CONTINUE CURSOR1 into :clobhv;
  - Consume :clobhv content
  - end loop
- Another way is to use FETCH … INTO DESCRIPTOR :SQLDA

Rename Index / Rename Column

Act.Right.Now.



RENAME INDEX/COLUMN

- Without having to drop and recreate the object
- Rename Column

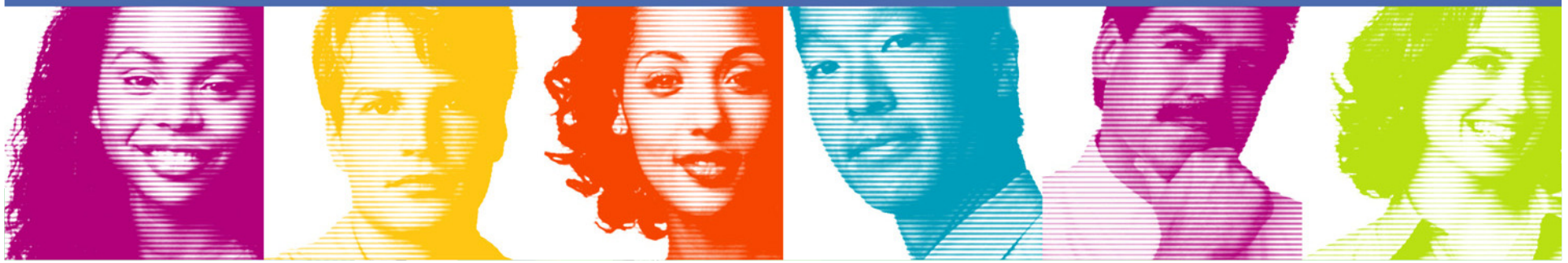
```
ALTER TABLE tb1
   RENAME COLUMN   old_columnname
     TO    new_columnname
```

- Rename Index

```
RENAME  INDEX/TABLE
     old_name  TO  new_name
```

Automatic Creation of Objects

Act.Right.Now.

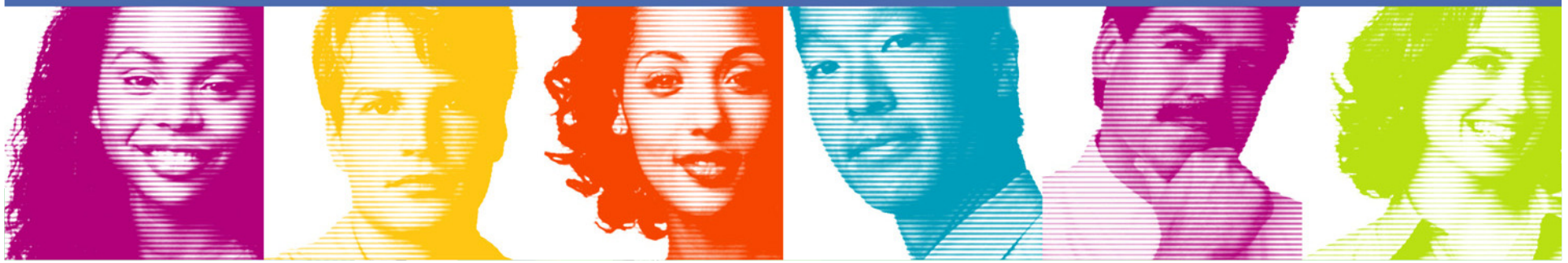


What's New?

- In V9, CREATE TABLE without specifying an associated table space and database
  - **Implicit Database**
    - DB2 creates an implicit database
    - Use DSN0001 to DSN60000 as naming convention for implicit created databases.
    - If DSN60000 is reached, DB2 wraps around and uses existing implicitly created databases.
    - Max # of database per DB2 subsystem has been increased from 32767 to 65271
  - **Implicit Table space**
    - Starting V9, the implicitly created table spaces will be segmented table spaces.
    - CM Mode
      - SEGSIZE 4, LOCKSIZE ROW
    - NFM Mode
      - Implicitly created table spaces are **Partition By Growth table spaces**.
      - SEGSIZE 4, DSSIZE 4G, MAXPARTITIONS 256, LOCKSIZE ROW, LOCKMZE SYSTEM

pureXML in DB2 9

Act.Right.Now.

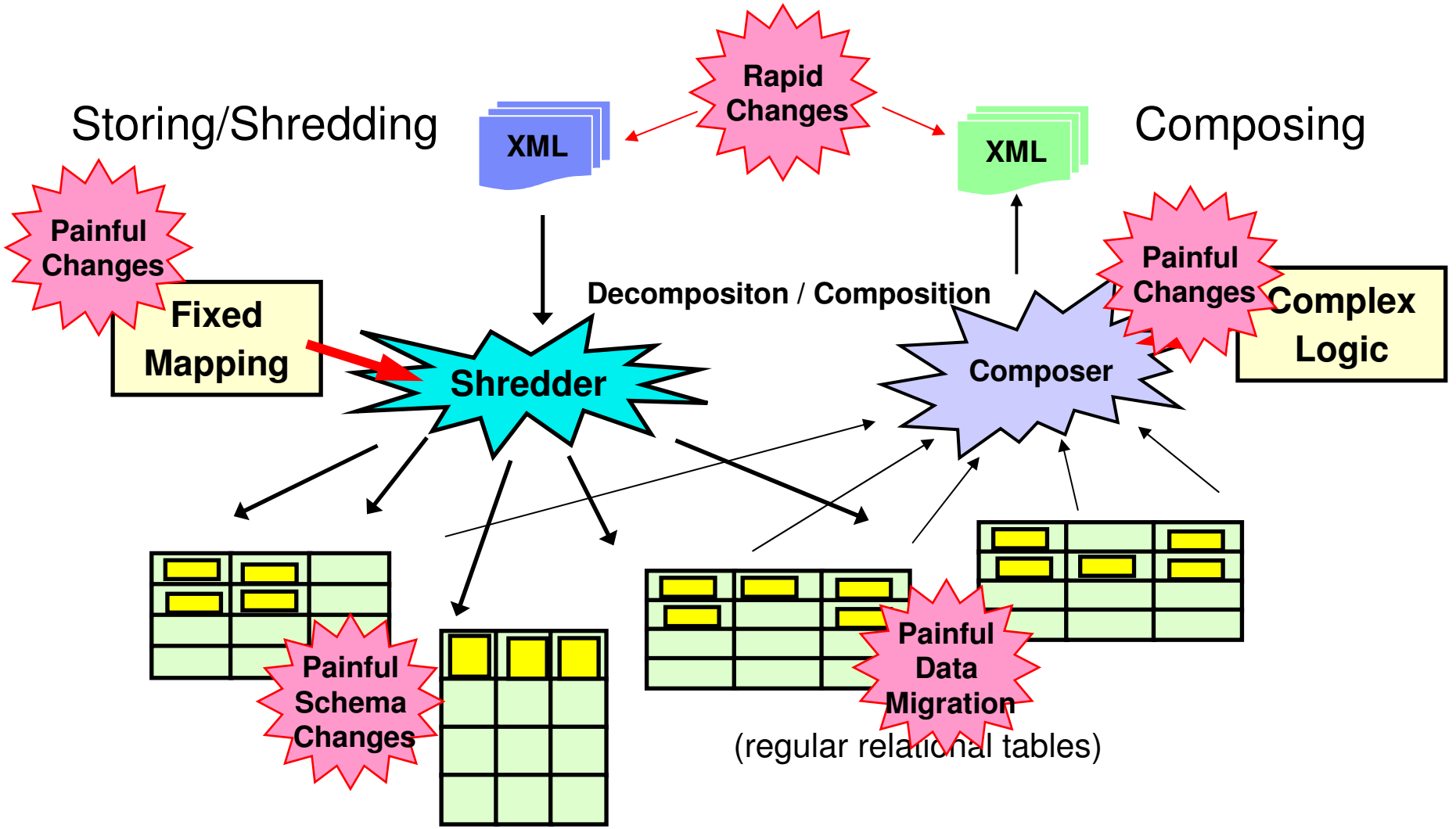


Example of V8 XML publishing functions

```
<Department name="Shipping">
   <emp>Lee</emp>
   <emp>Martin</emp>
   <emp>Oppenheimer</emp>
</Department>
```

```
SELECT XML2CLOB(
          XMLELEMENT(NAME "Department",
            XMLATTRIBUTES (e.dept AS "name" ),
              XMLAGG(XMLELEMENT(NAME "emp", e.lname)
                        ORDER BY e.lname  )
                     ) ) AS "dept_list"
FROM employees e
GROUP BY dept;
```

XML Data Processing before pureXML



pureXML in DB2 9

- SQL XML data type and native storage
- Designed specifically for XML
  - Supports XML hierarchical structure storage
  - Native operations and languages: XPath, SQL/XML, (XQuery in the future)
- Not transforming into relational
- Not using objects or nested tables
- Not using LOBs
- Integrated with relational engine, with all the utilities and tools support

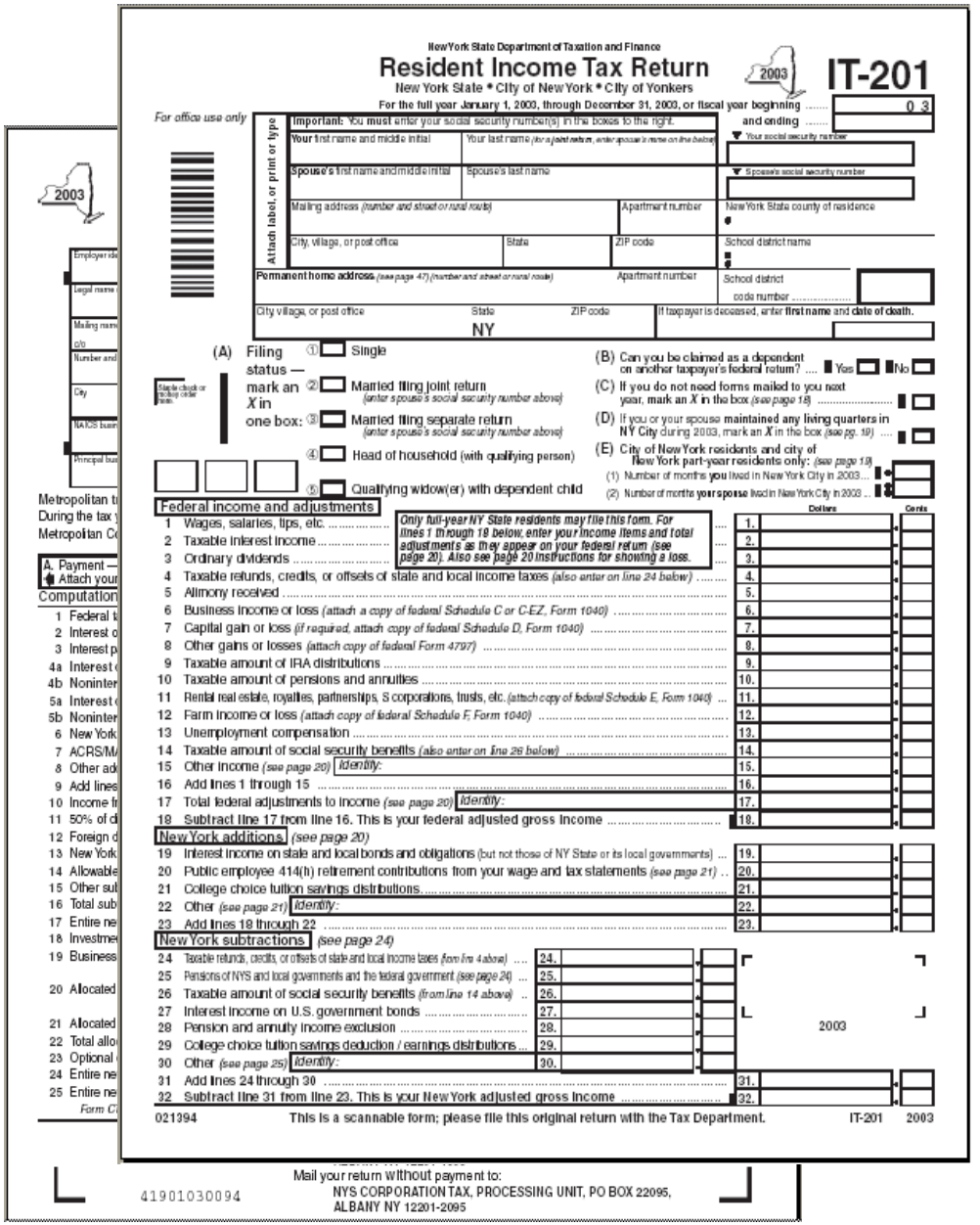
Example: Tax Forms

- Application
  - Processing & validating tax returns, payments, refunds
  - Corporate Tax, Personal Income Tax (PIT), Sales Tax
- Objectives
  - Move Tax processing off legacy systems
    - **Move to a more flexible, automated, extensible framework**
      Reduce cost & labor for implementing tax form changes
  - Increase performance. Improve straight-through processing from filing to refund/payment
- Typical current environment
  - Processing using manual and/or legacy systems
- This is an example of usage for Online Forms processing in general

Tax Forms

- Usually hundreds-thousands of different tax forms
  → **Schema Diversity**
- Typically not every field in a form is used
  → **Sparse Data**
- Many forms change every year
  → **Schema Evolution**

➔ A case for XML !



New York State Department of Taxation and Finance

Resident Income Tax Return

New York State * City of New York * City of Yonkers

2003 **IT-201**

For the full year January 1, 2003, through December 31, 2003, or fiscal year beginning ..... and ending .....

For office use only

Important: You must enter your social security number(s) in the boxes to the right.

Your first name and middle initial | Your last name | Your social security number

Spouse's first name and middle initial | Spouse's last name | Spouse's social security number

Attach label, or print or type

Mailing address (number and street or rural route) | Apartment number | New York State county of residence

City, village, or post office | State | ZIP code | School district name

Permanent home address (see page 47) (number and street or rural route) | Apartment number | School district code number

City, village, or post office | State: NY | ZIP code | If taxpayer is deceased, enter first name and date of death.

(A) Filing status — mark an X in one box: ① Single ② Married filing joint return (enter spouse's social security number above) ③ Married filing separate return (enter spouse's social security number above) ④ Head of household (with qualifying person) ⑤ Qualifying widow(er) with dependent child

(B) Can you be claimed as a dependent on another taxpayer's federal return? .... Yes No

(C) If you do not need forms mailed to you next year, mark an X in the box (see page 18) .....

(D) If you or your spouse maintained any living quarters in NY City during 2003, mark an X in the box (see pg. 19) ....

(E) City of New York residents and city of New York part-year residents only: (see page 19)
(1) Number of months you lived in New York City in 2003 ...
(2) Number of months your spouse lived in New York City in 2003 ...

Federal income and adjustments

Only full-year NY State residents may file this form. For lines 1 through 19 below, enter your income items and total adjustments as they appear on your federal return (see page 20). Also see page 20 instructions for showing a loss.

		Dollars	Cents
1	Wages, salaries, tips, etc.	1.	
2	Taxable interest income	2.	
3	Ordinary dividends	3.	
4	Taxable refunds, credits, or offsets of state and local income taxes (also enter on line 24 below)	4.	
5	Alimony received	5.	
6	Business income or loss (attach a copy of federal Schedule C or C-EZ, Form 1040)	6.	
7	Capital gain or loss (if required, attach copy of federal Schedule D, Form 1040)	7.	
8	Other gains or losses (attach copy of federal Form 4797)	8.	
9	Taxable amount of IRA distributions	9.	
10	Taxable amount of pensions and annuities	10.	
11	Rental real estate, royalties, partnerships, S corporations, trusts, etc. (attach copy of federal Schedule E, Form 1040)	11.	
12	Farm income or loss (attach copy of federal Schedule F, Form 1040)	12.	
13	Unemployment compensation	13.	
14	Taxable amount of social security benefits (also enter on line 26 below)	14.	
15	Other income (see page 20) Identify:	15.	
16	Add lines 1 through 15	16.	
17	Total federal adjustments to income (see page 20) Identify:	17.	
18	Subtract line 17 from line 16. This is your federal adjusted gross income	18.	

New York additions (see page 20)

19	Interest income on state and local bonds and obligations (but not those of NY State or its local governments)	19.	
20	Public employee 414(h) retirement contributions from your wage and tax statements (see page 21)	20.	
21	College choice tuition savings distributions	21.	
22	Other (see page 21) Identify:	22.	
23	Add lines 18 through 22	23.	

New York subtractions (see page 24)

24	Taxable refunds, credits, or offsets of state and local income taxes (from line 4 above)	24.	
25	Pensions of NYS and local governments and the federal government (see page 24)	25.	
26	Taxable amount of social security benefits (from line 14 above)	26.	
27	Interest income on U.S. government bonds	27.	
28	Pension and annuity income exclusion	28.	
29	College choice tuition savings deduction / earnings distributions	29.	
30	Other (see page 25) Identify:	30.	
31	Add lines 24 through 30	31.	
32	Subtract line 31 from line 23. This is your New York adjusted gross income	32.	

021394 This is a scannable form; please file this original return with the Tax Department. IT-201 2003

41901030094

Mail your return without payment to:
NYS CORPORATION TAX, PROCESSING UNIT, PO BOX 22095,
ALBANY NY 12201-2095

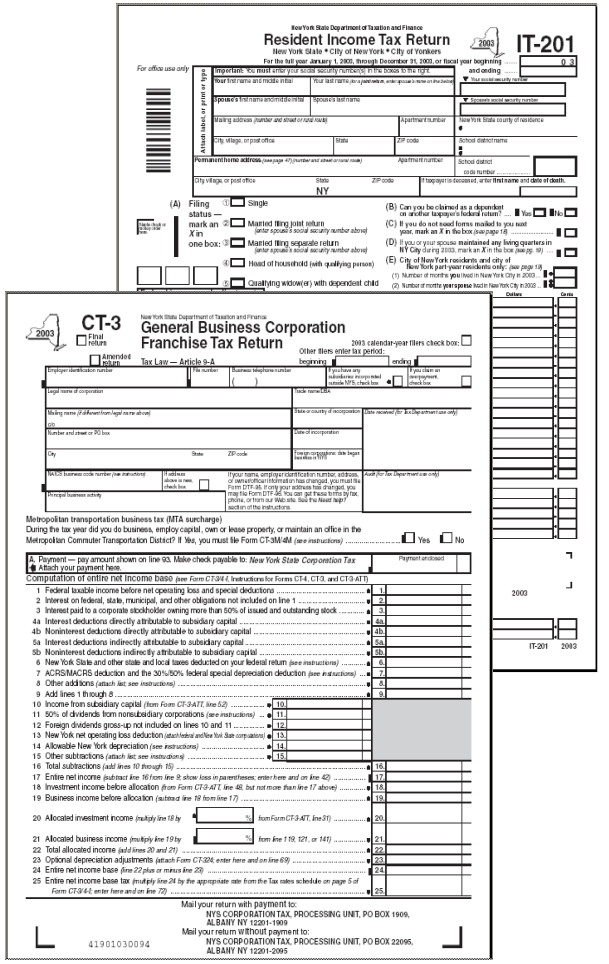
Typical Current Usage: Relational Database

- Solution 1: Each form has a different set of fields (schema)

  → Thousands of Tables … i.e. one per form ?

- Considered not feasible
  - Too many tables to maintain
  - Relational schema would deteriorate over time
  - Not sufficiently flexible and extensible

- Solution 2: Single table whose rows can store *any* form
  - 100s of generic columns … Ouch!



Generic columns → XML

Current relational storage, inefficient, anonymous columns, requires complex mappings in the application

col1	col2	col3	col4	col5	…	col1000
134	NULL	11/23/05	NULL	NULL		NULL
NULL	276	NULL	NULL	Yes	…	NULL
12	NULL	NULL	99.99	NULL	…	NULL
NULL	NULL	NULL	123.23	NULL	…	No

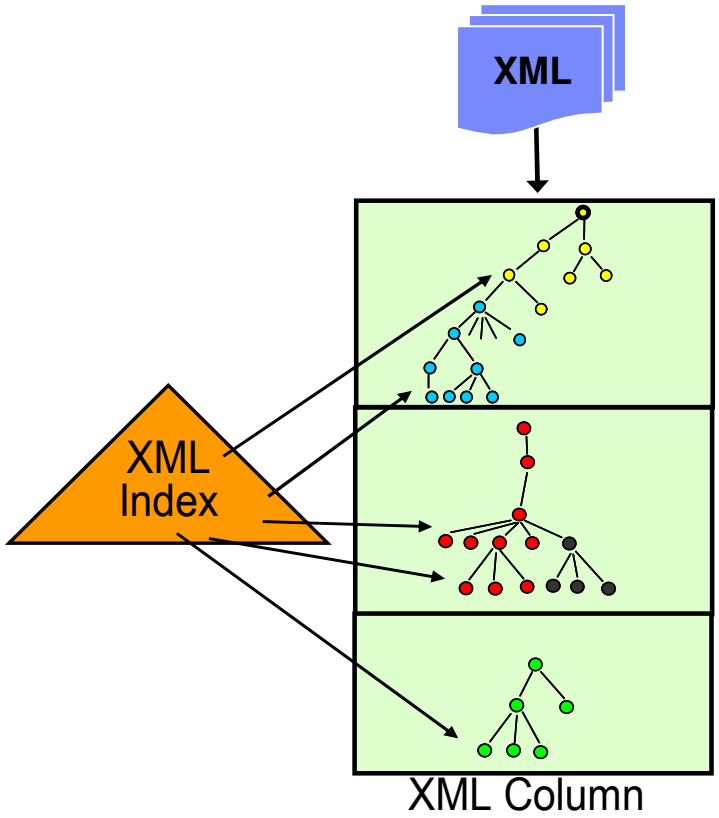
New XML format:

```
<form>
  <wages>134</wages>
  <date>11/23/05</date>
</form>
```

XML: Avoids sparsity. Proper data labeling. 2 columns, not 1000. Transformable. Extensible. Simplifies mapping.

What You Can Do with pureXML

- Create tables with XML columns
- Insert XML data, optionally validated against schemas
- Create indexes on XML data
- Efficiently search XML data
- Extract XML data
- Decompose XML data into relational data
- Construct XML documents from relational and XML data
- All the utilities and tools support for XML

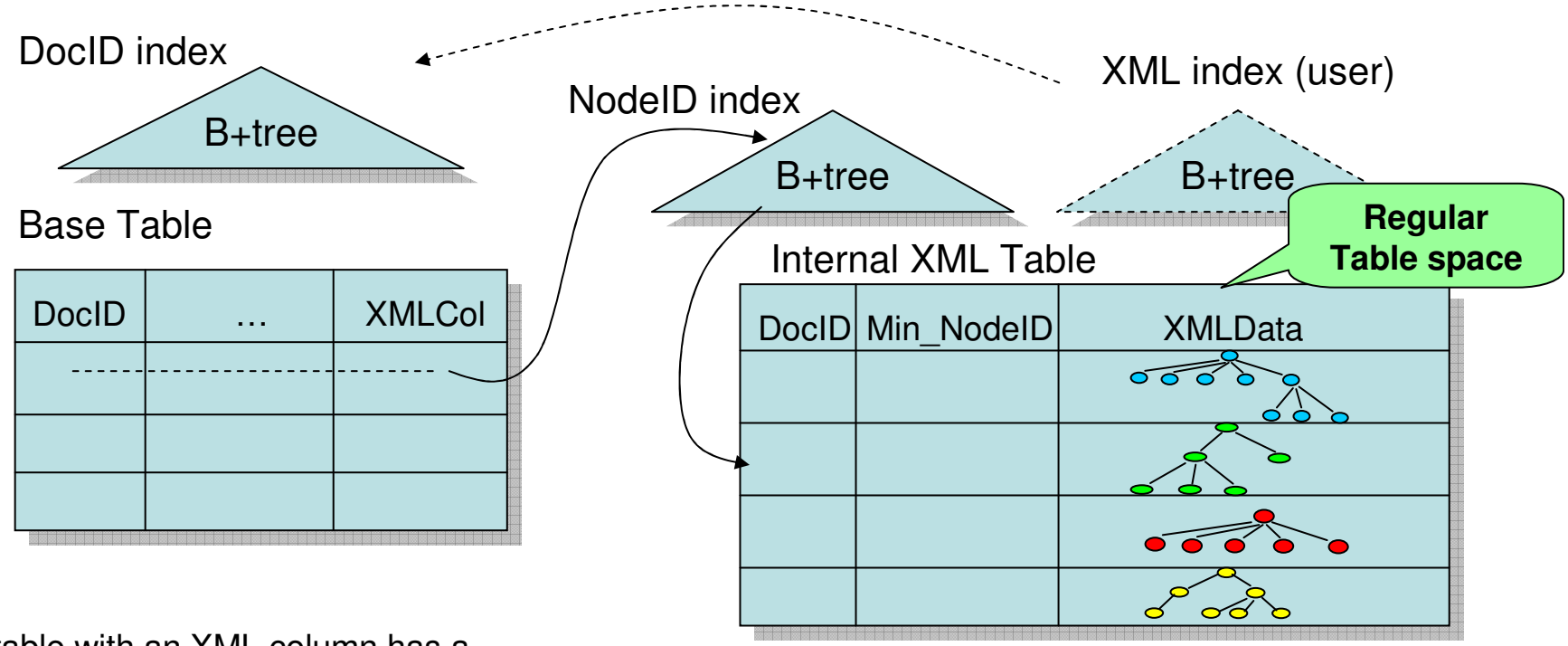


XML Type and DDL

```
CREATE TABLE PurchaseOrders (
  ponumber              varchar(10) not null,
  podate                date not null,
  status                char(1),
  XMLpo    xml);
     [or: IN MYDB.MYTS; ]
     [or: IN DATABASE MYDB; ]
     [or: IN MYTS; ]

CREATE TABLE PO LIKE PurchaseOrders;

CREATE VIEW ValidPurchaseOrders as
  SELECT ponumber, podate, XMLpo
     FROM PurchaseOrders
     WHERE status = 'A';

ALTER TABLE PurchaseOrders
    ADD revisedXMLpo xml;
```

- Hidden DocID column
- One DocID index
- Internal XML table (16K BP) for each XML column
- NodeID index
- No associated schema
- No length limit

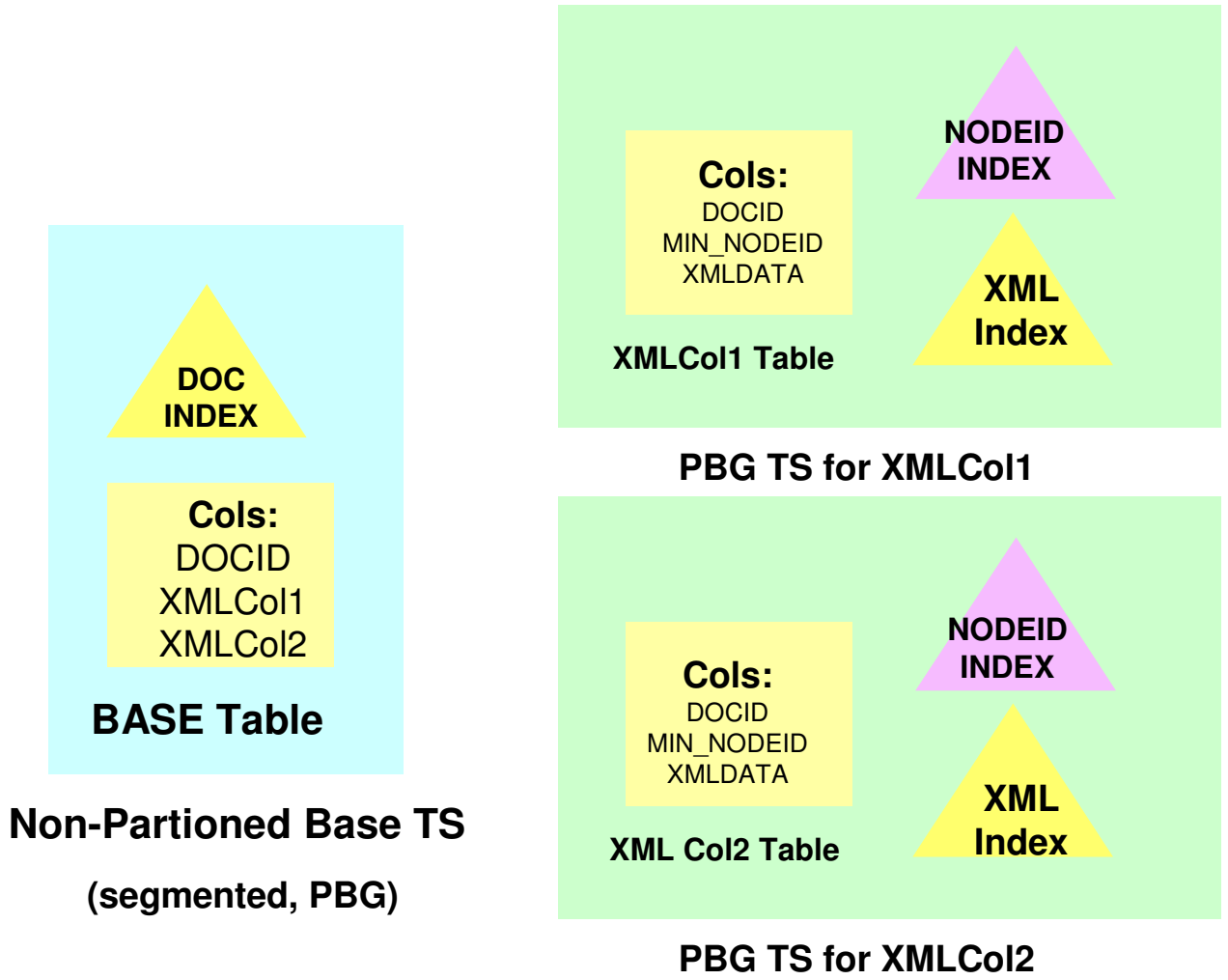
XML Storage on Mature Infrastructure



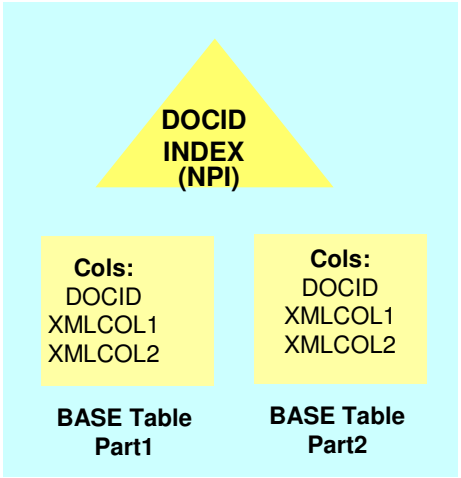
A table with an XML column has a DocID column, used to link from the base table to the XML table.
A DocID index is used for getting to base table rows from XPath value indexes.

Each XMLData column is a VARBINARY, containing a subtree or a sequence of subtrees, with context path.
Rows in XML table are freely movable, linked with a NodeID index.

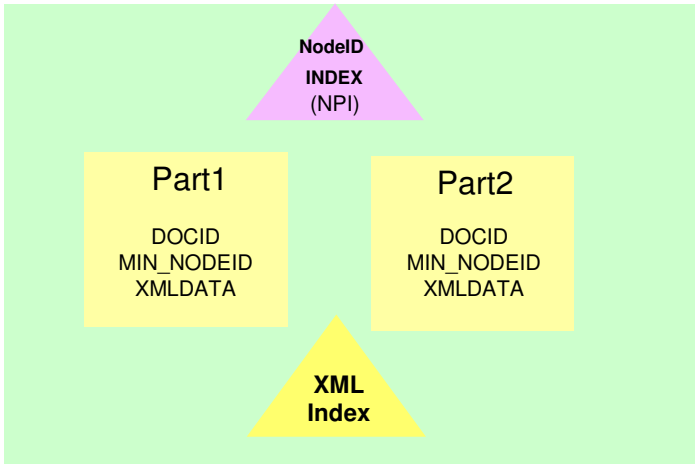
XML objects for non-partitioned base table



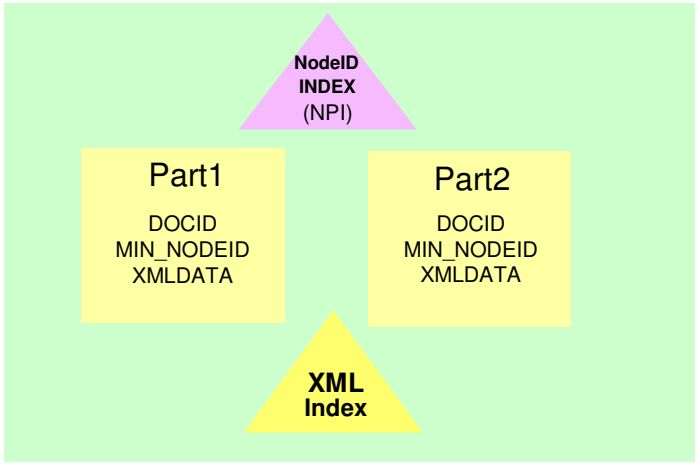
XML objects for partitioned base table



Partitioned Base TS
2 Parts, Table has 2
XML Columns



Part TS XMLCol1



Part TS XMLCol2

Manipulating XML Data

```
EXEC SQL BEGIN DECLARE SECTION;
      SQL TYPE IS XML AS CLOB(1M) xmlPo;
EXEC SQL END DECLARE SECTION;

INSERT INTO PurchaseOrders VALUES ('200300001',
            CURRENT DATE, 'A', :xmlPo);

INSERT INTO PurchaseOrders VALUES ('200300001',
            CURRENT DATE, 'A', CAST(? AS XML));

INSERT INTO PurchaseOrders VALUES('200300003', CURRENT DATE, 'A',
    XMLPARSE(DOCUMENT :vchar PRESERVE WHITESPACE) );

INSERT into PurchaseOrders VALUES( '200300004', CURRENT DATE, 'A',
    DSN_XMLValidate(:lobPo, 'SYSXSR.myPOSchema'));

UPDATE PurchaseOrders SET XMLpo = :XMLpo_revised
       WHERE ponumber = '12345';

DELETE FROM PurchaseOrders WHERE ponumber = '12345';
```

Host var of XML type

String literal is OK

Whole document replacement

Retrieving XML Data

- Simple select:

```
SELECT XMLpo INTO :xmlPo
FROM PurchaseOrders
WHERE ponumber = '200300001';
```

- Select with condition:

```
SELECT XMLPO
FROM PurchaseOrders
WHERE XMLEXISTS('//items/item[desc = "Shoe"]'   PASSING XMLpo);
```

- Extract from a document:

```
SELECT XMLQUERY('//items/item/quantity'   PASSING XMLpo)
FROM PurchaseOrders WHERE …;
```

Application Interfaces

- XML type is supported in
  - Java (JDBC, SQLJ), ODBC,
  - C/C++, COBOL, PL/I, Assembly
  - .NET
- Applications use:
  - XML as CLOB(n), XML as CLOB_FILE
  - XML as DBCLOB(n), XML as DBCLOB_FILE
  - XML as BLOB(n), XML as BLOB_FILE
  - All character or binary string types are supported

XML Indexes

- XPath value index: index values of elements or attributes inside a document.
- Index entries include:
  (key value, DocID, NodeID, RIDx)
- Support string (VARCHAR) or numeric (DECFLOAT) key type

```
CREATE INDEX ON
 PurchaseOrders(XMLPO) Generate
 Keys Using XMLPATTERN
 '/purchaseOrder/items/item/desc'
 as SQL VARCHAR(100);
```

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
  <shipTo country="US">
   <name>Alice Smith</name>
   . . .
  </shipTo>
  <billTo country="US">
   <name>Robert Smith</name>
   . . .
  </billTo>
  <comment>Hurry, my lawn is going wild!</comment>
  <items>
   <item partNum="872-AA">
    <desc>Lawnmower</desc>
    <quantity>1</quantity>
    <USPrice>148.95</USPrice>
    <comment>Confirm this is electric</comment>
   </item>
   <item partNum="926-AA">
    <desc>Baby Monitor</desc>
    <quantity>1</quantity>
    <USPrice>39.98</USPrice>
    <shipDate>2003-05-21</shipDate>
   </item>
  </items>
</purchaseOrder>
```

This index can be used for predicate:
XMLEXISTS('/purchaseOrder/items/item[desc = "Baby Monitor"]' passing XMLPO)

Utilities

- Enhanced to handle new XML type, XML tablespaces, and XML indexes
- CHECK DATA
- CHECK INDEX
- COPY INDEX
- COPY TABLESPACE
- COPYTOCOPY
- LISTDEF
- LOAD
- MERGECOPY
- QUIESCE TABLESPACESET
- REAL TIME STATISTICS
- REBUILD INDEX
- RECOVER INDEX
- RECOVER TABLESPACE
- REORG INDEX
- REORG TABLESPACE
- REPORT TABLESPACESET
- UNLOAD
- Basic RUNSTATS

Customer Experiences

- Insurance, financial, banking, government, healthcare, telecom, manufacturing, … (private list)
- References or public information:
  - ADP Netherlands: Payroll services (client XML data)
  - ZIVIT: Tax processing
  - Univar USA: Chemical Industry (CIDX contract)
  - GAD: Banking, XBRL & SEPA (financial report and payment)
  - Merrill Lynch: Finance
  - Temenos T24: universal banking application (object persistence)
- From LUW:
  - NY State: Tax processing
  - UCLA Health System: medical records
  - More at http://www.ibm.com/developerworks/wikis/display/db2xml/DB2+pureXML+Case+Studies