



Unleashing the Potential of Columnar Tables in Db2

Satya Krishnaswamy, IBM

Central Canada Db2 Users Group

Toronto

Please note :

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice and at IBM's sole discretion.

- Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.
- The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.
- The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.
- Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

AGENDA

Columnar Data Engine (CDE) in Db2

Storage and compression of Columnar Table

Recent improvements

What's coming

Creating a Column-Organized Table

```
CREATE TABLE sales_col (  
  c1 INTEGER NOT NULL,  
  c2 INTEGER,  
  . . .  
  PRIMARY KEY (c1) ) ORGANIZE BY COLUMN;
```

- If `dft_table_org = COLUMN`
 - `ORGANIZE BY COLUMN` is the default and can be omitted
 - Use `ORGANIZE BY ROW` to create row-organized tables
- `DB2_WORKLOAD=ANALYTICS` Sets Everything You Need

What you see in the Db2 catalog: TABLEORG

- Which tables are column-organized?
 - Column In syscat.tables: TABLEORG

```
SELECT tabname, tableorg, compression
FROM   syscat.tables
WHERE  tabname like 'SALES%';
```

TABNAME	TABLEORG	COMPRESSION
SALES_COL	C	
SALES_ROW	R	N

2 record(s) selected.

For column-organized tables, COMPRESSION is always blank because you cannot enable/disable compression.

Columnar storage in Db2 (conceptual)

Name	Age	Address	City	State	Zip Code
John Piconne	47	18 Main Street	Springfield	MA	01111
Susan Nakagawa	32	455 N. 1 st St.	San Jose	CA	95113
Sam Gerstner	55	911 Elm St.	Toledo	OH	43601
Chou Zhang	22	300 Grand Ave	Los Angeles	CA	90047
Mike Hernandez	43	404 Escuela St.	Los Angeles	CA	90033
Pamela Funk	29	166 Elk Road #47	Beaverton	OR	97075
Rick Washington	78	5661 Bloom St.	Raleigh	NC	27605
Ernesto Fry	35	8883 Loghorn Dr.	Tucson	AZ	85701
Whitney Samuels	80	14 California Blvd.	Pasadena	CA	91117
Carol Whitehead	61	1114 Apple Lane	Cupertino	CA	95014

Row-Organized Table Format

- Traditional approach: data stored in row format

John Piconne	47	18 Main Street	Springfield	MA	01111
Susan Nakagawa	32	455 N. 1 st St.	San Jose	CA	95113
Sam Gerstner	55	911 Elm St.	Toledo	OH	43601
Chou Zhang	22	300 Grand Ave	Los Angeles	CA	90047
Mike Hernandez	43	404 Escuela St.	Los Angeles	CA	90033
Pamela Funk	29	166 Elk Road #47	Beaverton	OR	97075
Rick Washington	78	5661 Bloom St.	Raleigh	NC	27605
Ernesto Fry	35	8883 Longhorn Dr.	Tucson	AZ	85701
Whitney Samuels	80	14 California Blvd.	Pasadena	CA	91117
Carol Whitehead	61	1114 Apple Lane	Cupertino	CA	95014

Page

Page

Page

Page

- Each page contains 1 or multiple rows (all columns)

Tuple Sequence Number (TSN)

TSN =
Tuple
Sequence
Number

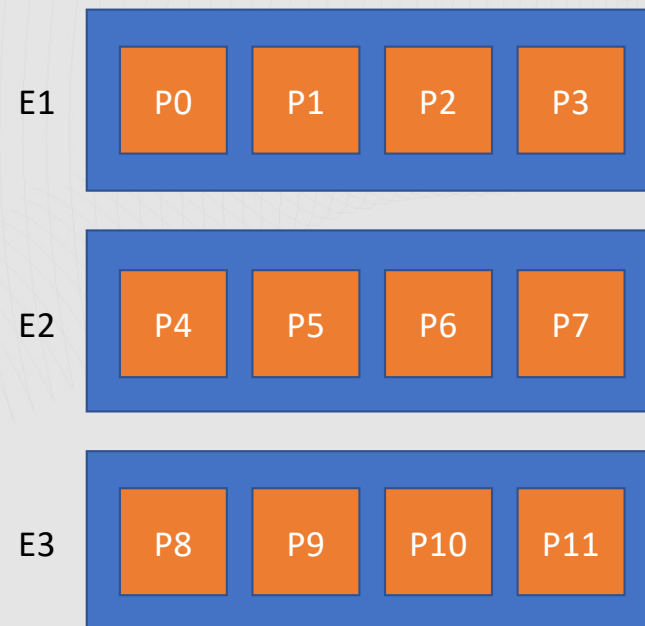
TSN	Column Group 1	Column Group 2	Column Group 3	Column Group 4	Column Group 5	Column Group 6
0	John Piconne	47	18 Main Street	Springfield	MA	01111
1	Susan Nakagawa	32	455 N. 1 st St.	San Jose	CA	95113
2	Sam Gerstner	55	911 Elm St.	Toledo	OH	43601
3	Chou Zhang	22	300 Grand Ave	Los Angeles	CA	90047
4	Mike Hernandez	43	404 Escuela St.	Los Angeles	CA	90033
5	Pamela Funk	29	166 Elk Road #47	Beaverton	OR	97075
6	Rick Washington	78	5661 Bloom St.	Raleigh	NC	27605
7	Ernesto Fry	35	8883 Longhorn Dr.	Tucson	AZ	85701
8	Whitney Samuels	80	14 California Blvd.	Pasadena	CA	91117
9	Carol Whitehead	61	1114 Apple Lane	Cupertino	CA	95014
10						
11						
...						

Annotations: A yellow arrow labeled "tuple" points to the row with TSN 1. A red arrow labeled "page" points to the row with TSN 3. A green arrow labeled "tuplelet" points to the cell containing "85701" (TSN 7, Column Group 6). A red arrow labeled "page" points to the row with TSN 9.

- Each **tuple** (row) in the table is assigned a **TSN**, which is similar to a Row ID.
- A **tuplelet** is the projection of a column group on a tuple
- TSNs are used to stitch tuplelets together during query processing
- The “start TSN” of each page is stored in the Page Map Index (PMI) for lookup of values by TSN

Extents and Pages

- Each column group has their own **data pages** to store column data
- Data pages are grouped into **extents**
- Extents, rather than individual pages, are the basic unit of data to be allocated and freed when required
- Typically, four pages will make up one extent



Column Groups and NULL values

- Each column in the table belongs to exactly one column group
- Currently, the only reason for a column group to hold multiple columns is if a column is nullable.
- The column in the table will be represented in the column group as a nullable column holding the data and an internal null indicator column.
 - The null indicator column contains 1-bit values indicating if the value for the corresponding column in the table is null.
- Null indicator values are either packed together with the data values in a single bank or stored in a separate bank within a region.

Null values example

Table (External Representation):

Name (NOT NULL)	Age	Address	City	State	Zip Code
John Piconne	NULL	18 Main Street	NULL	MA	01111
Susan Nakagawa	32	NULL	San Jose	NULL	95113

Table (Internal Representation):

Name	Age	NI (Age)	Address	NI (Address)	City	NI (City)	State	NI (State)	Zip Code	NI (Zip Code)
John Piconne	?	1	18 Main Street	0	?	1	MA	0	01111	0
Susan Nakagawa	32	0	?	1	San Jose	0	?	1	95113	0

? = Value that can be encoded to the least amount of bits

Synopsis Table

- Meta-data that describes which *ranges* of values exist in which parts of the user table

SYN130330165216275152_SALES_COL

TSNMIN	TSNMAX	S_DATEMIN	S_DATEMAX	...
0	1023	2005-03-01	2006-10-17	...
1024	2047	2006-08-25	2007-09-15	...
...				

TSN = Tuple Sequence Number

User table: SALES_COL

S_DATE	QTY	...
2005-03-01	176	...
2005-03-02	85	...
2005-03-02	267	
2005-03-04	231	
...		
...		
...		
...		
...		
...		

0

1023

1024

2047

- Enables Db2 to skip portions of table when scanning data during query
- Benefits from data clustering, loading pre-sorted data
- Predicate WHERE S_DATE = 2007-01-01 would skip first range
- Predicate WHERE S_DATE = 2006-09-12 would scan both ranges



Synopsis Table: Example

Base Table:

TSN	Name	Age
0	John Piconne	47
1	Susan Nakagawa	32
2	Sam Gerstner	55
3	Chou Zhang	22
4	Mike Hernandez	43
5	Pamela Funk	29
6	Rick Washington	78
7	Ernesto Fry	35
8	Whitney Samuels	80
9	Carol Whitehead	61

Synopsis Table (Synopsis tuple covers 5 base table tuples):

TSN MIN	TSN MAX	NAMEMIN	NAMEMAX	AGE MIN	AGE MAX
0	4	Chou Zhang	Susan Nakagawa	22	55
5	9	Carol Whitehead	Whitney Samuels	29	80

What you see in the Db2 catalog: Synopsis Tables

- Each columnar table has a corresponding synopsis table with a few exceptions
- Automatically created/maintained
- Queries use the synopsis table to determine if it can skip ranges of rows when evaluating predicates

```
SELECT tabschema, tabname, tableorg
FROM syscat.tables
WHERE tableorg = 'C';
```

TABSCHEMA	TABNAME	TABLEORG
MNICOLA	SALES_COL	C
SYSIBM	SYN130330165216275152_SALES_COL	C

```
2 record(s) selected.
```

When Data is Encoded vs. Unencoded

- Most columns have a mix of both encoded and unencoded values.

Encoded	Unencoded
<ul style="list-style-type: none">• A column value is stored encoded if a dictionary entry can be used to encode that data• Dictionary entry can be used• Internal columns like NULL indicator columns, which are encoded using 1 bit• DECIMAL columns, minus coding if precision ≤ 18 (imposed by 64-bit code size limit)	<ul style="list-style-type: none">• A value is stored unencoded if it cannot be represented in the dictionary• Values that occur infrequently (e.g., value occurs once, inefficient to create a dictionary value)• Values that were pruned from the histogram due to memory constraints• Values that were INSERTed/LOADed after dictionary creation, not covered by existing dictionary (static)

Encoding differs from **compression**, a value can be unencoded and still be compressed with ..., but the value needs to be decompressed before queries can be performed on it. This is why we refer to pure dictionary encoded and encoding types as *actionable* compression.

Columnar Compression

- Columnar compression dictionaries are used to compress data in a column of a columnar table by mapping repeated byte patterns to much smaller symbols, which then replace the longer byte patterns in the table.

- **Frequency compression**

- Most common values use fewest bits

- **Encoding Schemes**

- Pure dictionary coding
 - Prefix coding
 - Minus coding

Since all data in a column are of the same data type, we can **improve the compression ratio** by exploiting the skew in data distribution.

Column compression differs from row compression in that we **map values to dictionary codes**.

Benefits

- **Save disk space** by reducing total amount of **buffer pool** pages needed to store data.
- **Faster queries**, a compressed table need fewer I/O operations to access the same amount of data.

Column-level Dictionaries are Static

- Once created, **column-level dictionaries are never updated**

- REORG does not rebuild column-level dictionaries
- Row organized tables can use REORG to rebuild the dictionary
- The user must unLOAD and reLOAD the table to rebuild the dictionary



Update Column-level
dictionaries

Page compression reduces the need to rebuild column dictionaries

- New values not covered by the initial column-level dictionary can still be compressed by the page-level dictionary
- This reduces deteriorating the compression ratio over time
- When a page fills up, decide whether to do page compression



Page Compression

Best Practices – Enable All Optional BLU Storage Enhancements

Feature	Description	Release	Db2	Db2 Warehouse
Page-based String Compression Type 1	Improves compression for high cardinality string columns with repeating portions of strings that are not encoded by existing compression algorithms	11.5.4	Need Registry Variable to Enable Registry Variable (2)	Enabled by Default
Page-Based String Compression Type 2	Improved compression when strings (within a page) contain 16 or less unique characters, works well for hex, numeric items like phone numbers, dates, dollar values when stored as strings	11.5.4	Need Registry Variable to Enable Registry Variable (2)	Enabled by Default
Deferred Synopsis Tuple Creation for Small Base Tables	Reduces synopsis table storage consumption for small base tables	11.5.4	Need Registry Variable to Enable Registry Variable (3)	Enabled by Default

Best Practices – Enable All Optional BLU Storage Enhancements

Feature	Description	Release	Db2	Db2 Warehouse
Reorg Table Recompress Enhancement	Improves performance of Reorg Table Recompress and applies page-based string compression during Reorg Table Recompress	11.5.5	Enabled by Default	Enabled by Default
Trickle Feed Insert Enhancement	Speeds up trickle feed insertion, reduces the memory footprint and size of small tables.	11.5.6	Not available	Enabled by Default
		11.5.7	Need Registry Variable to Enable Registry Variable (1)	Enabled by Default
LOAD Utility Enhancement	Improves overall LOAD processing, and also if the previously mentioned string compression algorithms are enabled allows LOAD to use them	11.5.8	Enabled by Default	Enabled by Default

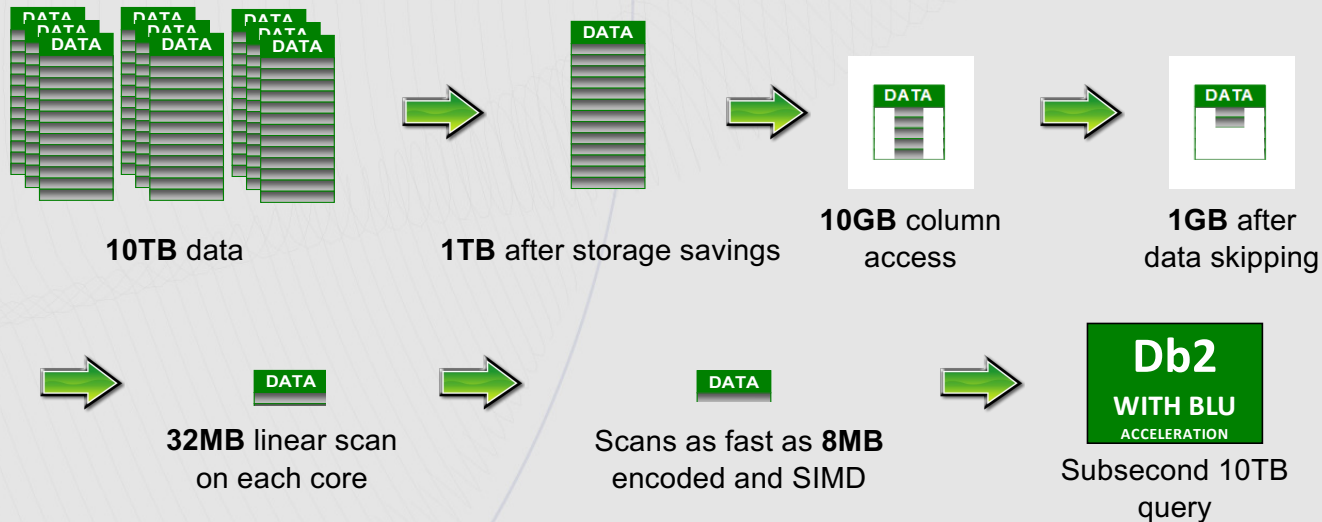
7 Big Ideas

- Simple to Implement and Use (Load & Go)
- Data Remains Compressed During Evaluation
- Multiply the Power of the CPU
- Core-Friendly Parallelism
- Work performed directly on columns
- Scan-Friendly Memory Caching
- Data skipping

7 Big Ideas: How Db2 with BLU Acceleration Helps

~Sub second 10TB query – An Optimistic Illustration

- The system – 32 cores, 10TB table with 100 columns, 10 years of data
- The query: `SELECT COUNT(*) from MYTABLE where YEAR = '2010'`
- The optimistic result: sub second 10TB query! Each CPU core examines the equivalent of just 8MB of data



Thank You

Speaker: Satya Krishnaswamy
Company: IBM
Email Address: satya.ksr@us.ibm.com