IMPROVING ETL PERFORMANCE

Calisto Zuzarte

IBM

Tridex : 02 Dec 2021 9:30 am

Agenda

- General ETL Considerations
  - Data Distribution, Statistics, Compression
- DML Considerations
  - LOAD/INSERT, DELETE, UPDATE, MERGE
- Performance Objects
  - Indexes, Replicated Tables, Materialized Query Tables

General ETL Considerations

Data Distribution, Statistics, Compression

Data Distribution

Data Distribution (1/2)

- CREATE TABLE … **DISTRIBUTE BY <column(s)>** …
  - User Specifies the distribution key
- CREATE TABLE … **DISTRIBUTE BY RANDOM** …
  - Distribution key is based on a hidden column automatically populated by a unique value generator
- CREATE TABLE … <**no distribute by clause**>
  - Db2 chooses a distribution key
  - If no Primary Key or unique constraint exists, three columns will be selected

Data Distribution (2/2)

- Very Large Tables (Fact Tables)
  - Collocate with the largest commonly joined (dimension) table
  - If all dimensions are small choose one or more common GROUP BY columns
- Medium to Large Tables
  - If Primary Key present, choose the Primary Key
  - If no Primary Key, choose the common join column to the largest table
- Small Tables
  - RANDOM distribution

Statistics

Statistics

ROW ORIENTED TABLES

- Auto-RUNSTATS OFF by default
- Real Time Statistics is available
- Indexes are more commonly defined and have additional statistics

COLUMN ORIENTED TABLES

- Auto-RUNSTATS is ON in Warehouses configured for column tables
- Real Time Statistics (Db2 11.5.6)
- Indexes not commonly defined hence need Auto-Column Group Statistics

Statistics Collection During ETL (1/4)

- Automatic RUNSTATS kicks in every 2 hours
  - It may be to late if it is a temporary table used during ETL
  - The Real Time Statistics feature added for Column-oriented tables
- Configuring Automatic RUNSTATS / Real Time Statistics:

```
Automatic table maintenance                (AUTO_TBL_MAINT) = ON
  Automatic runstats                         (AUTO_RUNSTATS) = ON
    Real-time statistics                   (AUTO_STMT_STATS) = ON
    Statistical views                    (AUTO_STATS_VIEWS) = ON
    Automatic sampling                       (AUTO_SAMPLING) = ON
    Automatic column group statistics  (AUTO_CG_STATS) = ON
```

Statistics Collection During ETL (2/4)

- Best practice for temporary ETL tables
  - If not using RTS, Collect RUNSTATS immediately AFTER the new data is loaded.
  - For large temporary tables, Use a low sampling rate for speed
  - SYSTEM (page) sampling is faster than BERNOULLI (row) sampling
- Example RUNSTATS statement

```
CALL SYSPROC.ADMIN_CMD ('RUNSTATS ON TABLE <schema.tablename>
WITH DISTRIBUTION ON ALL COLUMNS
AND INDEXES ALL TABLESAMPLE SYSTEM(10)');
```

Statistics Collection During ETL (3/4)

- Best Practice For RUNSTATS sampling
  - For tables that will be used during ETL or subsequent queries an appropriate sample size as recommended below may be used

```
CALL SYSPROC.ADMIN_CMD ('RUNSTATS ON TABLE
<schema.tablename>
WITH DISTRIBUTION ON ALL COLUMNS AND INDEXES ALL
TABLESAMPLE SYSTEM(N)
```

```
Very roughly, for up to 10 Million rows, use N = 25
Between 10 Million and 100 Million rows, use N = 20
Between 100 Million and 1 Billion rows,  use N = 10
More than 1 Billion rows,                use N = 5
```

Statistics Collection During ETL (4/4)

- Best Practice For RUNSTATS Column Group Statistics (CGS)
  - The Auto-CGS setting is recommended
  - CGS are advanced statistics not collected by default
  - Exploited for **multiple local predicates** / **multiple join predicates**

```
CALL SYSPROC.ADMIN_CMD ('RUNSTATS ON TABLE SCHEMA.T1
ON ALL COLUMNS AND COLUMNS ((C1, C3), (C4, C7))
WITH DISTRIBUTION AND INDEXES ALL TABLESAMPLE SYSTEM(20)
SET PROFILE');
```

COMPRESSION

Compression (1/3)

ROW ORIENTED TABLES

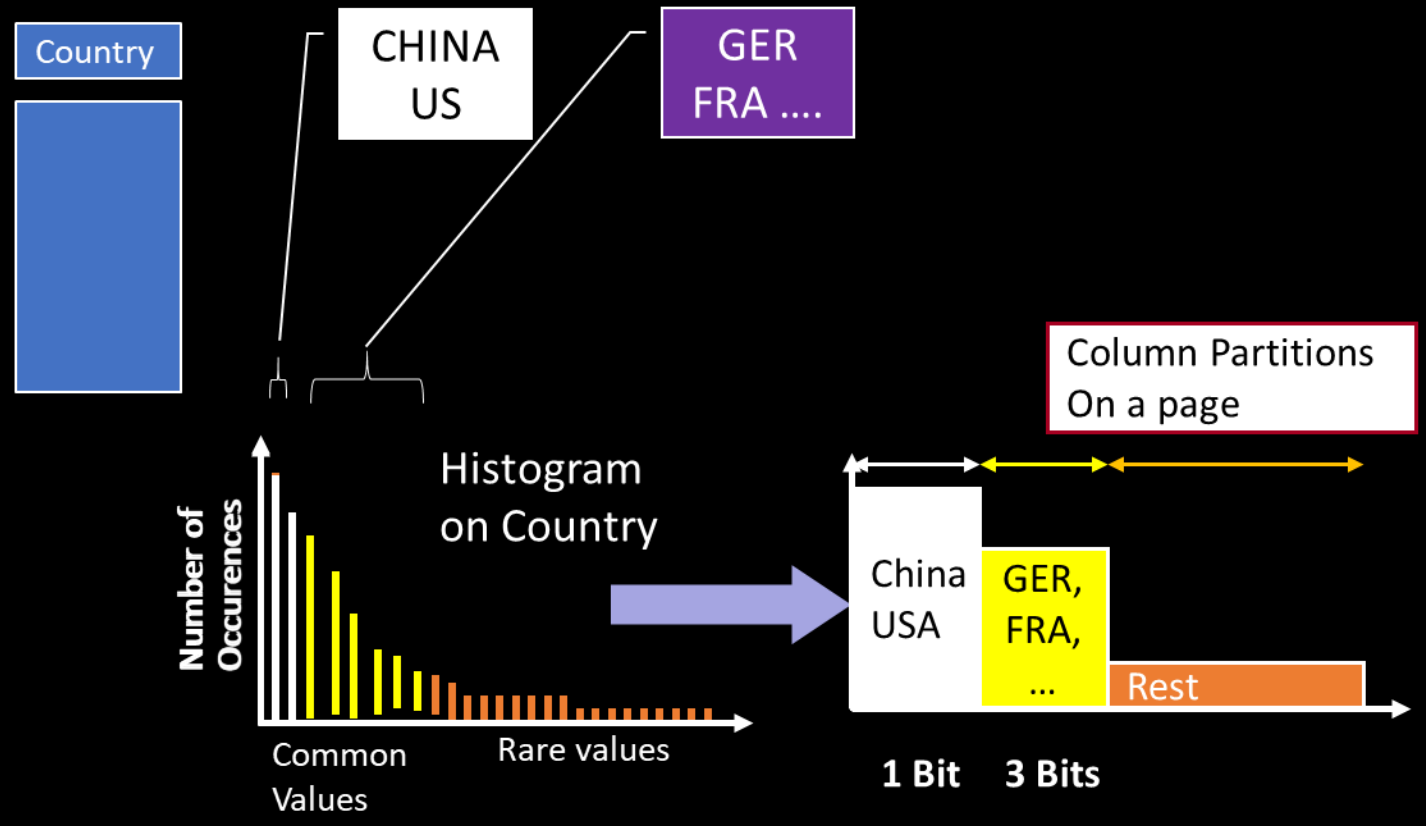
- Compression OFF by default
- Not recommended when CPU bound
- Data uncompressed when read
- Dictionary per table/range partition
- Additional Page Compression
- Partial column or multi-column compression

COLUMN ORIENTED TABLES

- Always compressed
- Operations exploit compressed data
- Dictionary per column
- Additional Page compression
- Order preserving frequency-based
- Compressed Row Indexes used.

Compression (2/3)

- Order Preserving Frequency Based Compression
- Dictionary Compression
- Page Compression



Compression (3/3)

- Best Practices for good compression in column-oriented tables
  - https://www.ibm.com/support/producthub/db2/docs/content/SSEPGG_11.5.0/com.ibm.db2.luw.common.doc/doc/c_bp_compress_blu.html
  - Paper that talks about
    - How to determine how well tables are compressed
    - How to determine how effective the column dictionaries are
    - How to rebuild and recompress data
    - How to maintain a table to keep a good compression ratio

DML

DML

ROW ORIENTED TABLES

- Full row in a single page
- DML not parallelized
- UPDATE is done in place
- Full logging

COLUMN ORIENTED TABLES

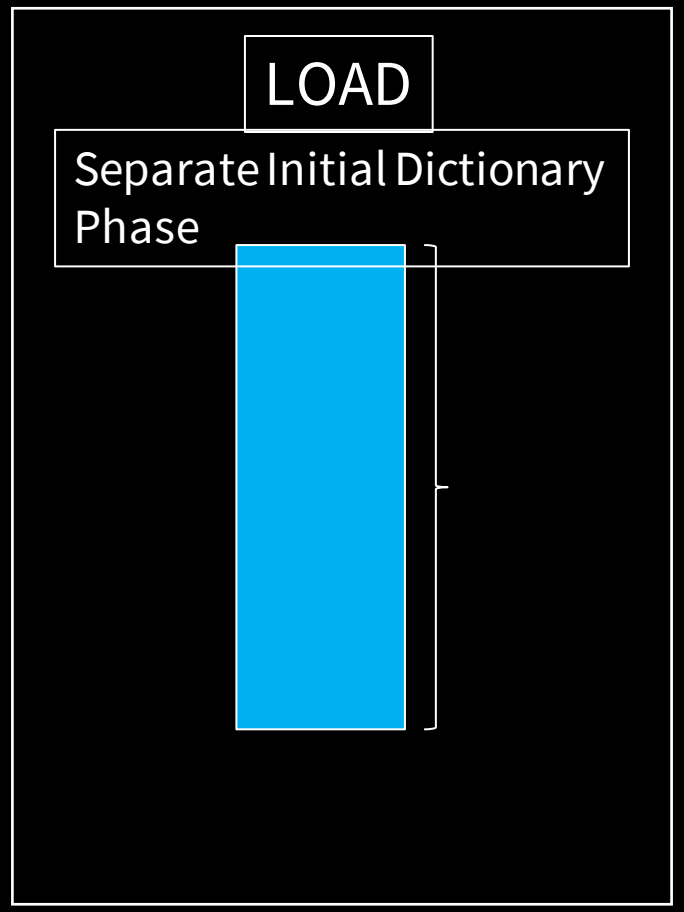
- Column data stored separately
- DML (not MERGE) is parallelized
- UPDATE uses DELETE + INSERT
- Reduced logging (95% less)
- 11.5.6: Improved trickle feed

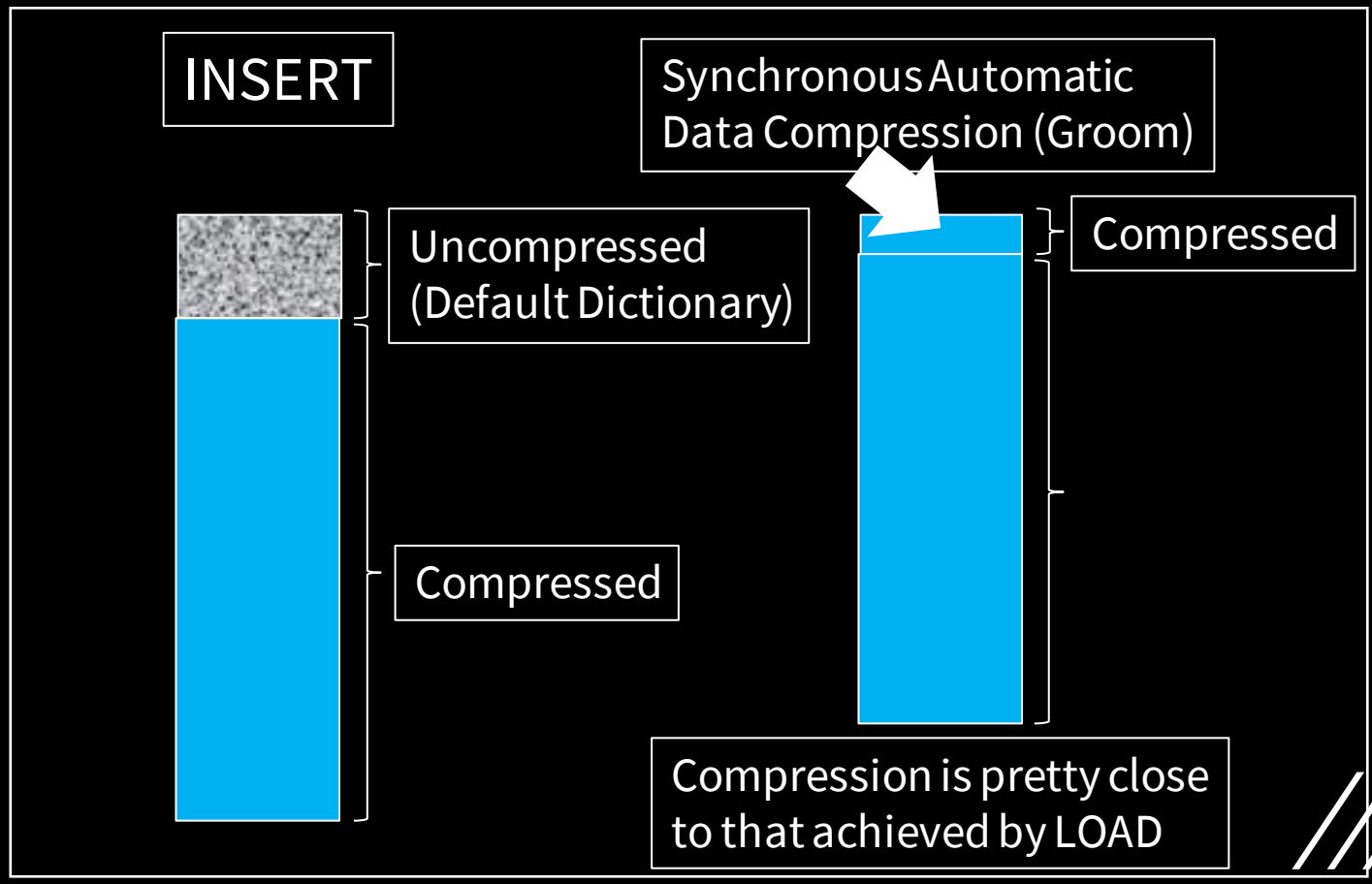
LOAD / INSERT

LOAD

- Fine grained Parallelism
- Ability to collect statistics automatically
- Can load data from external sources in various formats
- Typically invoked without logging (some index logging possible)
- Provides the best Compression rebuilding a table with RESETDICTIONARYONLY

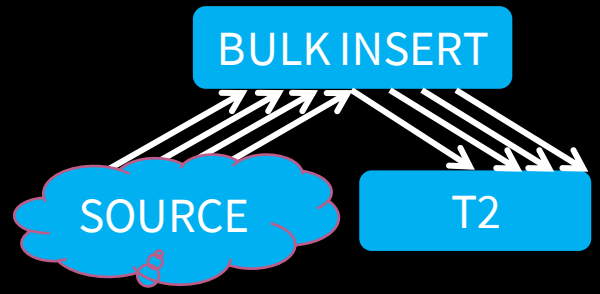
LOAD v/s INSERT - Compression





LOAD v/s INSERT - Performance

- INSERT is the preferred vehicle to insert data because in general with parallel formatting, parallel INSERT and highly reduced logging, it is faster than LOAD for column-oriented tables

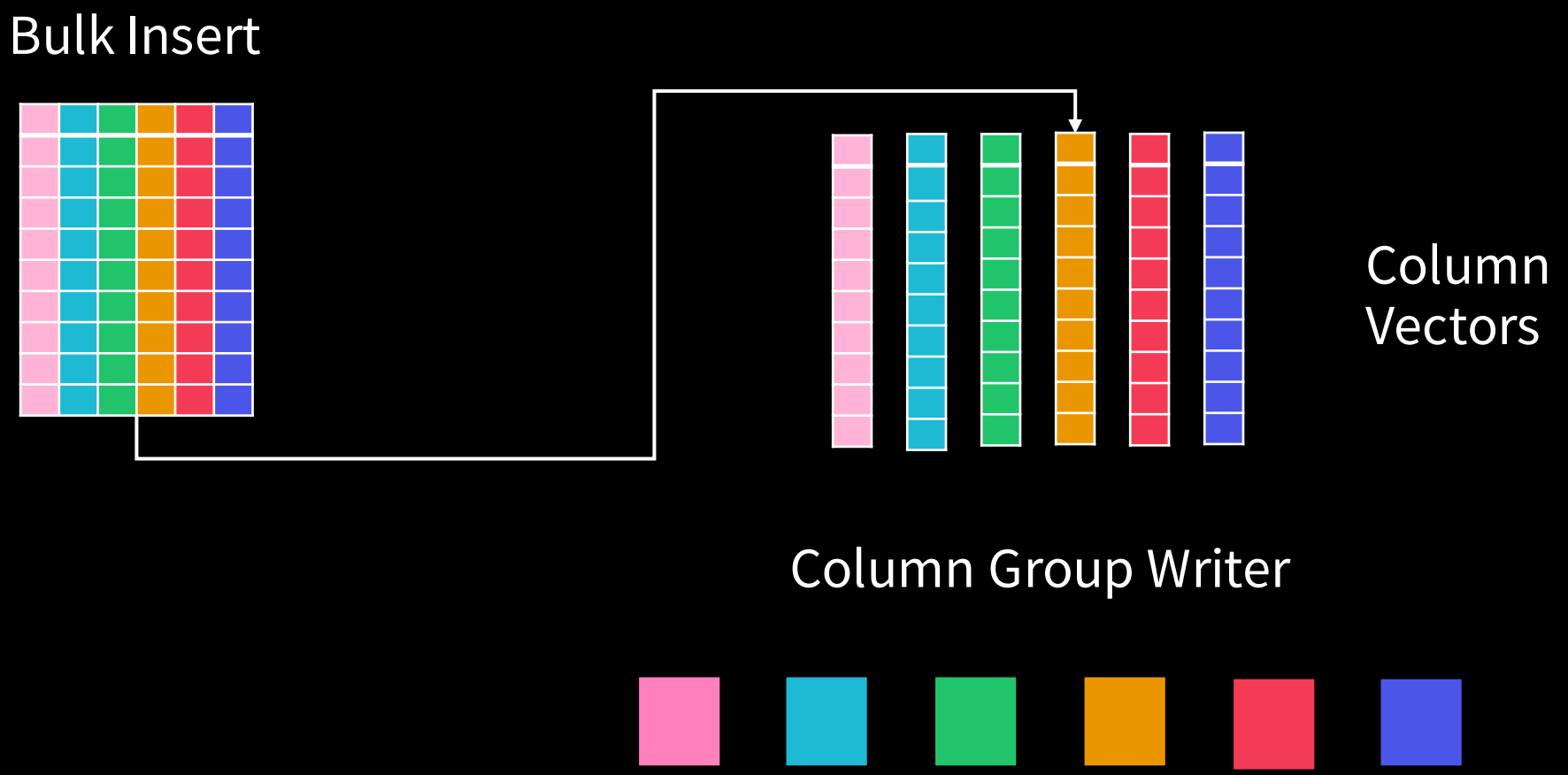


- The main advantage is the ability to avoid table locking in a concurrent environment allowing for suitable trickle feed loading of data.

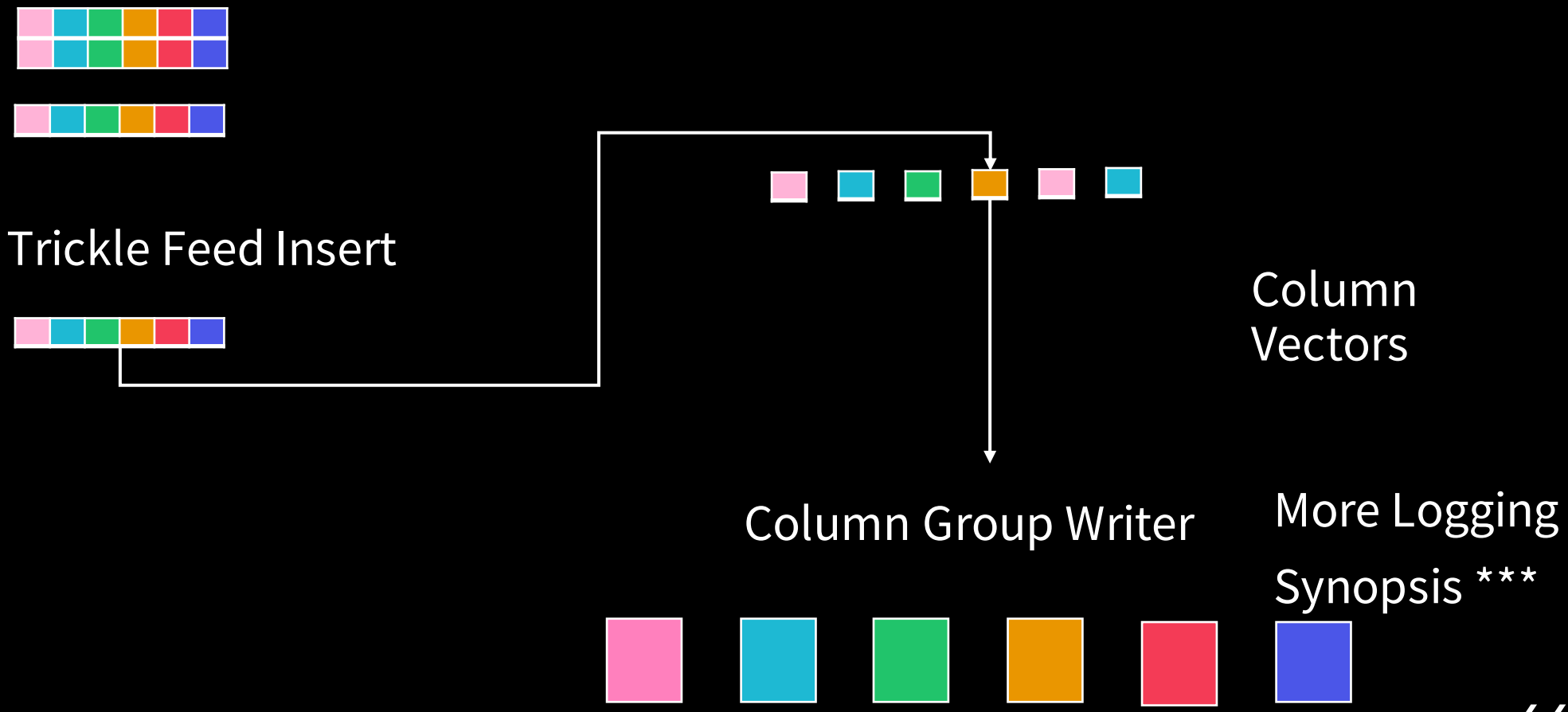
INSERT – Column-Oriented Tables

- Parallelized
- Vectorized bulk INSERTs
- Synopsis optimization
- Improved Automatic Dictionary Compression (ADC) in 11.5
  - Vectorized ADC
  - Larger amount of data used to build the dictionaries
  - Initial uncompressed data is compressed automatically
- Highly reduced logging compared to row-oriented table INSERT

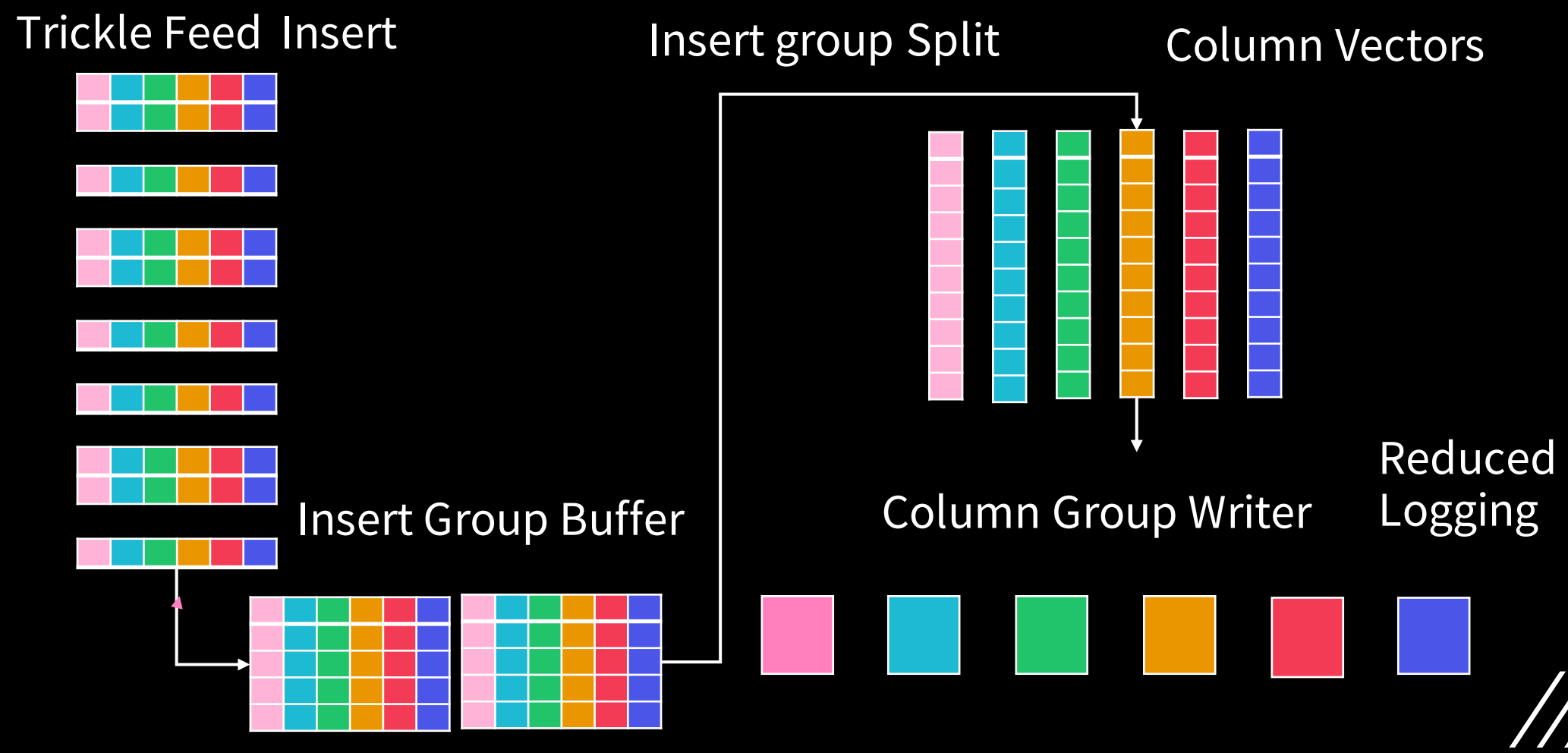
Bulk Insert



Trickle Feed INSERT before Db2 11.5.6



Trickle Feed INSERT in Db2 11.5.6



Improved Trickle Feed INSERT – 11.5.6

- Improved INSERT of a small number of rows
  - Initially inserted into “column group buffers”
  - When appropriate, tail buffers are split into column vectors
- Benefits
  - Reduced storage space for small tables
  - Significant reduction in log space usage
    (~ 50% - 75% with INTEGER columns)
  - Reduction in bufferpool dirty pages
  - Modest performance improvement

DML – Trickle Feed

11.5.5 / 11.5.6	ROW + Key index	COLUMN (No index)
Bulk INSERT	★★★	★★★★★
Bulk DELETE	★★★	★★★★★
Bulk UPDATE	★★★	★★★

11.5.5	ROW + Key index	COLUMN (No Index)
Trickle Feed INSERT	★★★	★★★
Trickle Feed DELETE	★★★	★★★
Trickle Feed UPDATE	★★★	★★★

11.5.6	ROW + Key index	COLUMN (No Index)
Trickle Feed INSERT	★★★	★★★
Trickle Feed DELETE	★★★	★★★
Trickle Feed UPDATE	★★★	★★★

- To enable Trickle Feed in Db2 11.5.6
  - db2set DB2_COL_INSERT_GROUPS=YES
  - Changes structures on disk so no fallback
- If you are OK with no version fallback you could benefit from better compression with the following
  - db2set DB2_COL_STRING_COMPRESSION= "UNENCODED_STRING:TRUE"

INSERT – Clustering Recommendations

- Appropriate columns to cluster on
  - Highly filtering local or join predicate columns
  - Datetime columns often used in predicates are ideal if these columns are loaded in datetime order

- How best to cluster
  - Small INSERTs < 1 M rows : INSERT …. SELECT ….. ORDER BY C1, C2
  - Large INSERTs need to split into ranges of the leading clustering column

DELETE

DELETE Optimizations

- Delete is parallelized in a column engine
- Deletes on Column oriented tables generally involve less I/O
- Compact logging compared to row-oriented tables
  - Orders of magnitude less I/O than row-oriented tables
- Single Row DELETE uses a fully qualified unique index if present
  - Multiple rows do not use indexes

DELETE Space Usage

- With logical deletion data pages are not modified on delete
- Space for the row is consumed until space reclaim is performed via
  - Automatic table maintenance (Enabled with DB2_WORKLOAD=ANALYTICS)
  - Manual invocation
    - REORG TABLE … RECLAIM EXTENTS
    - ALTER TABLESPACE … REDUCE
- Extents are only reclaimed if they have been fully deleted

TRUNCATE v/s DELETE v/s DROP/CREATE

- DROP TABLE / CREATE TABLE is common
  - Typical scenario and avoids keeping unused tables around
  - Dictionaries need to be re-built
- DELETE
  - There is an extra step to mark the rows as deleted.
  - Auto Table maintenance or REORG needs to be done to free extents
  - Dictionaries are not rebuilt when new data is loaded
- TRUNCATE
  - Similar to DELETE, all rows with more control to release or re-use storage
  - Recommend using the IMMEDIATE option and as the first statement in the transaction for ETL temporary tables where ROLLBACK is not required.

UPDATE

UPDATE processing

- UPDATE processing is decomposed to DELETE + INSERT
- Tables easily get un-clustered with UPDATE
- Less logging than Row tables (No Before + After images)
- The effect of Indexes
  - Optimized single row updates with a fully qualified unique index
  - Some parallelism is lost since updates to the index are serialized
  - Indexes are updated with UPDATE even with no change in the key value

UPDATE Optimization Tip 1 (Hack ☺)

- UPDATE T1 SET C2 = 5
  - Recall UPDATE ➔ DELETE + INSERT
  - The decomposed UPDATE needs to read all the columns to do the INSERT
  - This may be done using random I/O when processing each row

- TIP :
  - UPDATE T1 SET C2 = 5, **C3 = C3, C4 = C4**
  - Redundant SET clause for C3, C4 and C5 reads the column pages sequentially

UPDATE Optimization Tip 2

- Assume C1 is the distribution column
  - UPDATE T1 SET **C1 = ?,** C2 = CURRENT DATE WHERE **C1 = ?**

- Performance TIP :
  - If C1 is set to the same parameter value as used in the WHERE clause, remove C1 = ? In the SET clause
  - UPDATE T1 SET ~~**C1 = ?,**~~ C2 = CURRENT DATE WHERE C1 = ?
  - Db2 does not know if the value will be changed and must take care of moving the row to a different database partition

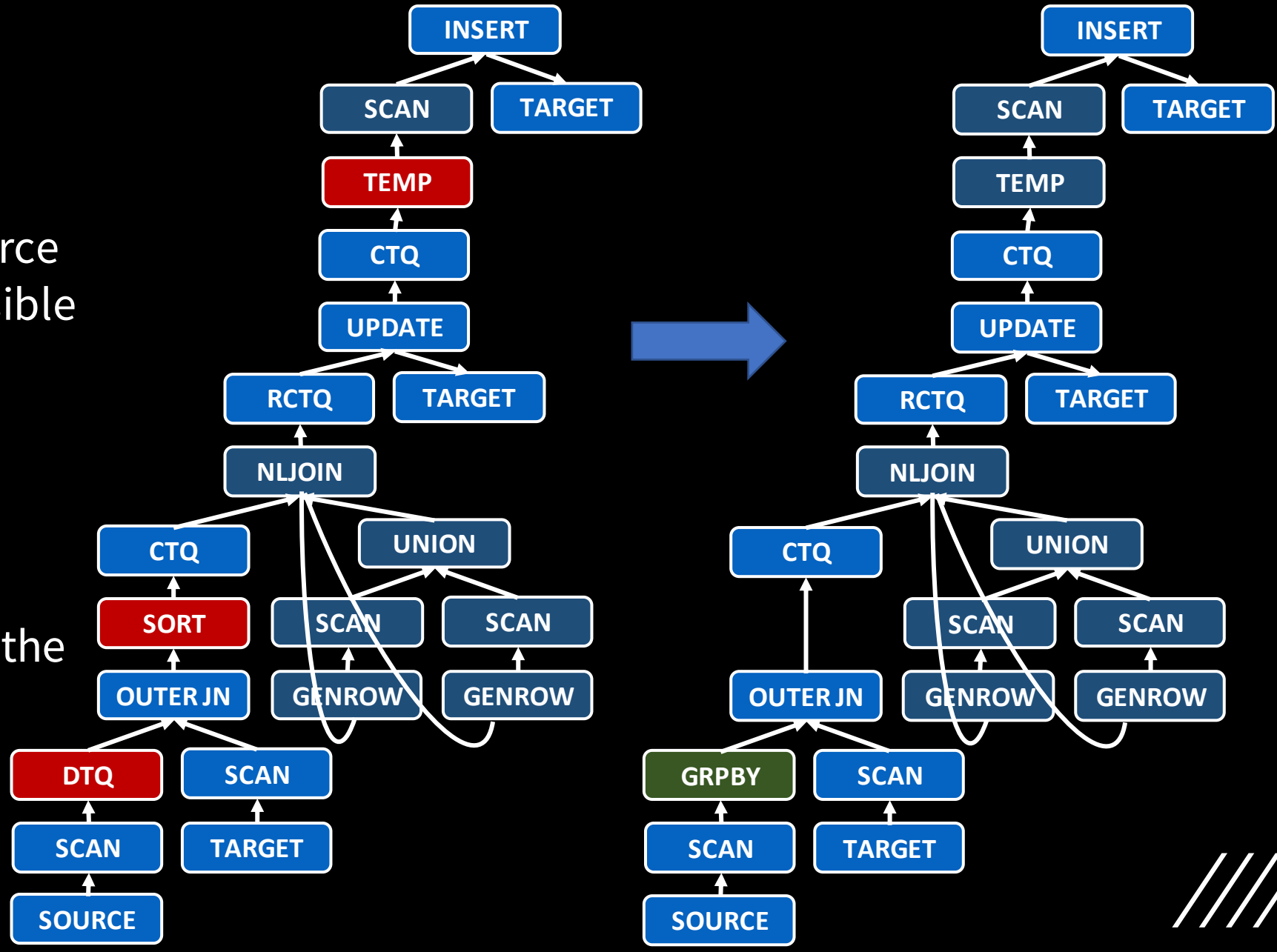
MERGE

Merge Recommendations

- Collocate the source table and the target table if possible
- Put a UNIQUE constraint on the source key (or GROUP BY & MAX)
  - This eliminates an OLAP function that needs a SORT to flag duplicate errors
- Minimize the TEMP size by specifying the DML that deals with most of the rows first.
- For MERGE parallelism, invoke each DML as separate statements

Merge

- Collocate the Source and Target if possible
- Avoid a SORT by defining a Unique source key
- One way to avoid the TEMP is if INSERT and UPDATE are done separately



Notes on DML

- INSERT is faster than LOAD
- A single row UPDATE or DELETE uses a fully qualified unique index
  - Multi-row UPDATE or DELETE does not exploit the index
- Indexes currently limit INSERT, UPDATE or DELETE parallelism

Performance Objects

Indexes

- For the most part the synopsis table is very effective with a table that is well clustered on common predicate columns

- Indexes may be considered for
  - Enforcing uniqueness
  - Single row UPDATE and DELETE
  - Queries that have highly filtering predicates (<= 1% of the table)

Replicated Tables (1/2)

- Benefits
  - Avoids traffic between database partitions
  - Reduces the number of agents required to process the query
- Only user maintained replicated tables (no system maintenance support).
  - Ideal if the data does not change during query workload and the user can maintain the replicated tables during ETL
- Ideal candidates to consider replicated tables
  - Small dimension tables
  - A vertical subset of the commonly referenced columns in medium dimension tables

Replicated Tables (2/2)

- Creating a replicated table SCH.T1_REPL of base table SCH.T1

```
CREATE TABLE SCH.T1_REPL AS (SELECT * FROM SCH.T1)
    DATA INITIALLY DEFERRED REFRESH DEFERRED  MAINTAINED BY USER REPLICATED IN USERSPACE1;

SET INTEGRITY FOR SCH.T1_REPL  ALL IMMEDIATE UNCHECKED;

INSERT INTO SCH.T1_REPL  (SELECT * FROM SCH.T1);

RUNSTATS ON TABLE SCH.T1_REPL WITH DISTRIBUTION ON ALL COLUMNS AND INDEXES ALL SET PROFILE;
```

- To exploit the replicated table, set these when queries are run

```
SET CURRENT REFRESH AGE ANY;
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION USER;
```

Materialized Query Tables (1/2)

- Benefits
  - Exploits pre-computed portions of a query for improved query performance
- Only user-maintained MQTs are supported
  - No “immediate” or “deferred” system maintenance support.
  - Ideal if the data does not change during query workload and the user can maintain the MQTs during ETL
- Ideal candidate MQTs
  - Common expensive joins with relatively small results
  - Aggregations over just the fact table grouping on subsets of the foreign keys

Materialized Query Tables (2/2)

- Creating a replicated table SCH.T1_REPL of table SCH.T1

```
CREATE TABLE SCH.MQT_YEAR AS
(   SELECT COUNTRY_KEY, ORGANIZATION_KEY,
           SUM(CURRENT_YEAR_REV) AS SUM_CY_REV,
           SUM(CURRENT_YEAR_COST) AS SUM_CY_COST
   FROM   SCH.REVENUE_COST_PERIOD
   GROUP BY COUNTRY_KEY, ORGANIZATION_KEY )
DATA INITIALLY DEFERRED REFRESH DEFERRED
MAINTAINED BY USER
DISTRBUTE BY (COUNTRY_KEY,ORGANIZATION_KEY)
ORGANIZE BY COLUMN IN USERSPACE1;
```

```
SET INTEGRITY FOR SCH.MQT_YEAR ALL IMMEDIATE UNCHECKED;

INSERT INTO SCH.MQT_YEAR
(   SELECT COUNTRY_KEY, ORGANIZATION_KEY,
           SUM(CURRENT_YEAR_REV) AS SUM_CY_REV,
           SUM(CURRENT_YEAR_COST) AS SUM_CY_COST
   FROM   SCH.REVENUE_COST_PERIOD
   GROUP BY COUNTRY_KEY, ORGANIZATION_KEY );

RUNSTATS ON TABLE SCH.MQT_YEAR WITH DISTRIBUTION ON ALL
COLUMNS;
```

- To use the MQT, set the following when the queries are run

```
SET CURRENT REFRESH AGE ANY;
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION USER;
```

Summary

- Various recommendations given to improve DML performance
- Trickle feed INSERT performance is significantly improved in Db2 11.5.6
  - Consider enabling this feature if version fallback is not an option
- If appropriate, consider User Maintained Replicated Tables and MQTs as part of the ETL to improve query workload performance

Speaker: Calisto Zuzarte
Company: IBM
Email Address: calisto@ca.ibm.com