Five Years Out

CL451 DB2 11 BLU Acceleration Implementation and Use

(Course code CL451)

Course Guide

ERC 1.0

Course Guide

CL451 DB2 11 BLU Acceleration Implementation and Use

Course code CL451 ERC 1.0

IBM Training

July 2017

Contents

Course overview

Preface overview

The course was created to provide a learning option for students that need to prepare for using the DB2 BLU Acceleration facilities of DB2 11.1 for Linux, UNIX and Windows systems.

The DB2 10.5 Fix Pack 4, referred to as Cancun, added support for Shadow tables, a new type of Materialized Query Table, and also Column-organized User Maintained MQT tables. One lecture unit describes these features. A demonstration allows students to implement and experiment with these functions.

With DB2 11.1, BLU Acceleration can be used in a clustered multiple database partition DB2 environment. This course includes a lecture and demonstration that allows students to create a set of column-organized tables from an existing set of row-organized tables and execute and analyze the performance of BLU Acceleration in a MPP database.

The lab demonstrations are performed using DB2 LUW 11.1 for Linux.

Intended audience

This is an advanced course for DB2 LUW experienced database administrators who support DB2 for UNIX, Windows, and Linux databases and want to learn more about the DB2 with BLU acceleration capabilities in DB2 11.1. These skills can also be utilized to support cloud based databases using DB2 on Cloud or IBM dashDB.

Topics covered

Topics covered in this course include:

- BLU Acceleration concepts
- BLU Acceleration implementation and use
- Implementing Shadow tables and BLU MQTs
- DB2 BLU MPP support

Course prerequisites

Participants should have completed one of the following courses:

- DB2 11.1 Administration Workshop for Linux (CL206)
- DB2 11.1 Quickstart for Experienced Relational DBAs (CL486)

Document conventions

Conventions used in this guide follow Microsoft Windows application standards, where applicable. As well, the following conventions are observed:

- **Bold**: Bold style is used in demonstration and exercise step-by-step solutions to indicate a user interface element that is actively selected or text that must be typed by the participant.
- *Italic*: Used to reference book titles.
- CAPITALIZATION: All file names, table names, column names, and folder names appear in this guide exactly as they appear in the application.
  To keep capitalization consistent with this guide, type text exactly as shown.

Demonstrations

Demonstrations format

Demonstrations are designed to provide hands on experience with some of the key concepts and skills covered by the lecture content.

The demonstrations for this course include the following:

- Creating a new DB2 database with the DB2_WORKLOAD set to enable BLU Acceleration default options.
- Creating and loading a set of row-organized and column-organized tables using the same source data to compare various storage and performance characteristics.
- Convert a row-organized table into a column-organized table using the ADMIN_TABLE_MOVE procedure.
- Compare access plans used for row-organized table access to column-organized tables.
- Creating several not enforced unique and foreign key constraints on the column-organized tables to improve access efficiency.
- Define user maintained column-organized materialized query tables and shadow tables to reduce access costs for analytics query processing.
- Load a set of tables from an existing DB2 database into a multiple partition DB2 database using column-organized tables.
- Utilize the DB2 explain tool to review access plans for joining column-organized tables that span multiple database partitions.
- Modify the distribution key for column-organized tables to enable collocated join processing with multi-partition tables.
- Create a replicated materialized query table to support table join processing in DB2 MPP databases.

The demonstrations utilize DB2 LUW 11.1 on Linux.

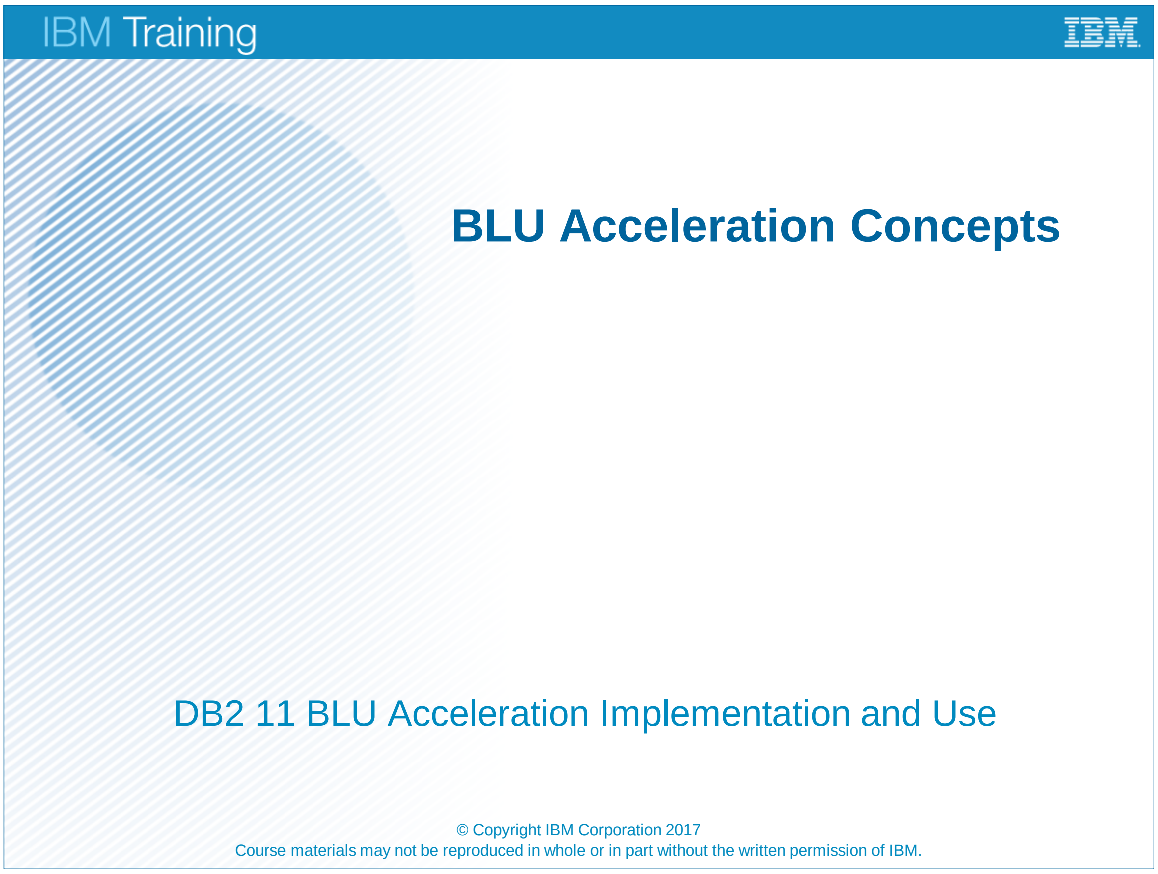
Additional training resources

- Visit IBM Analytics Product Training and Certification on the IBM website for details on:
  - Instructor-led training in a classroom or online
  - Self-paced training that fits your needs and schedule
  - Comprehensive curricula and training paths that help you identify the courses that are right for you
  - IBM Analytics Certification program
  - Other resources that will enhance your success with IBM Analytics Software
- For the URL relevant to your training requirements outlined above, bookmark:
  - Information Management portfolio: http://www-01.ibm.com/software/data/education/
  - Predictive and BI/Performance Management/Risk portfolio: http://www-01.ibm.com/software/analytics/training-and-certification/

IBM product help

Help type	When to use	Location
Task-oriented	You are working in the product and you need specific task-oriented help.	*IBM Product* - Help link
Books for Printing (.pdf)	You want to use search engines to find information. You can then print out selected pages, a section, or the whole book. Use Step-by-Step online books (.pdf) if you want to know how to complete a task but prefer to read about it in a book. The Step-by-Step online books contain the same information as the online help, but the method of presentation is different.	Start/Programs/*IBM Product*/Documentation
IBM on the Web	You want to access any of the following:	
	• IBM - Training and Certification	• http://www-01.ibm.com/software/analytics/training-and-certification/
	• Online support	• http://www-947.ibm.com/support/entry/portal/Overview/Software
	• IBM Web site	• http://www.ibm.com

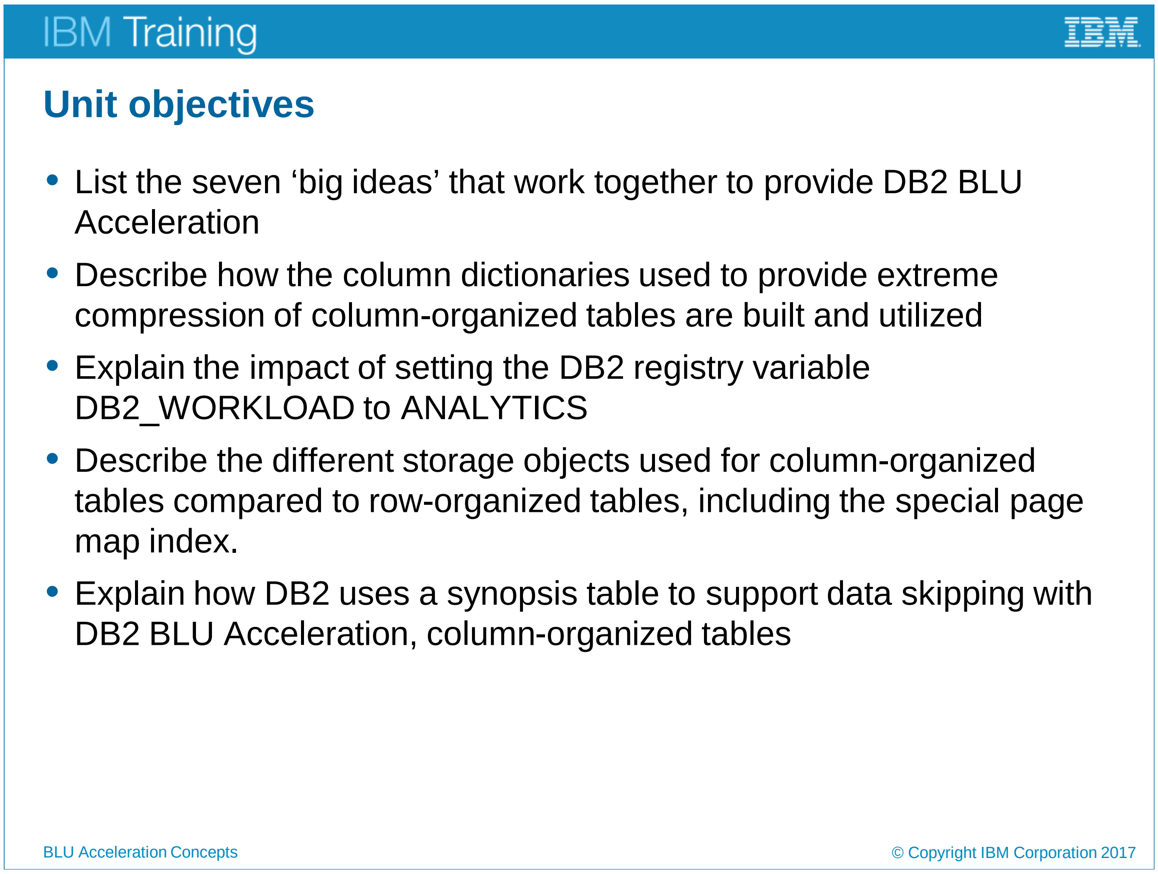
Unit 1 BLU Acceleration Concepts



IBM Training

BLU Acceleration Concepts

DB2 11 BLU Acceleration Implementation and Use



IBM Training

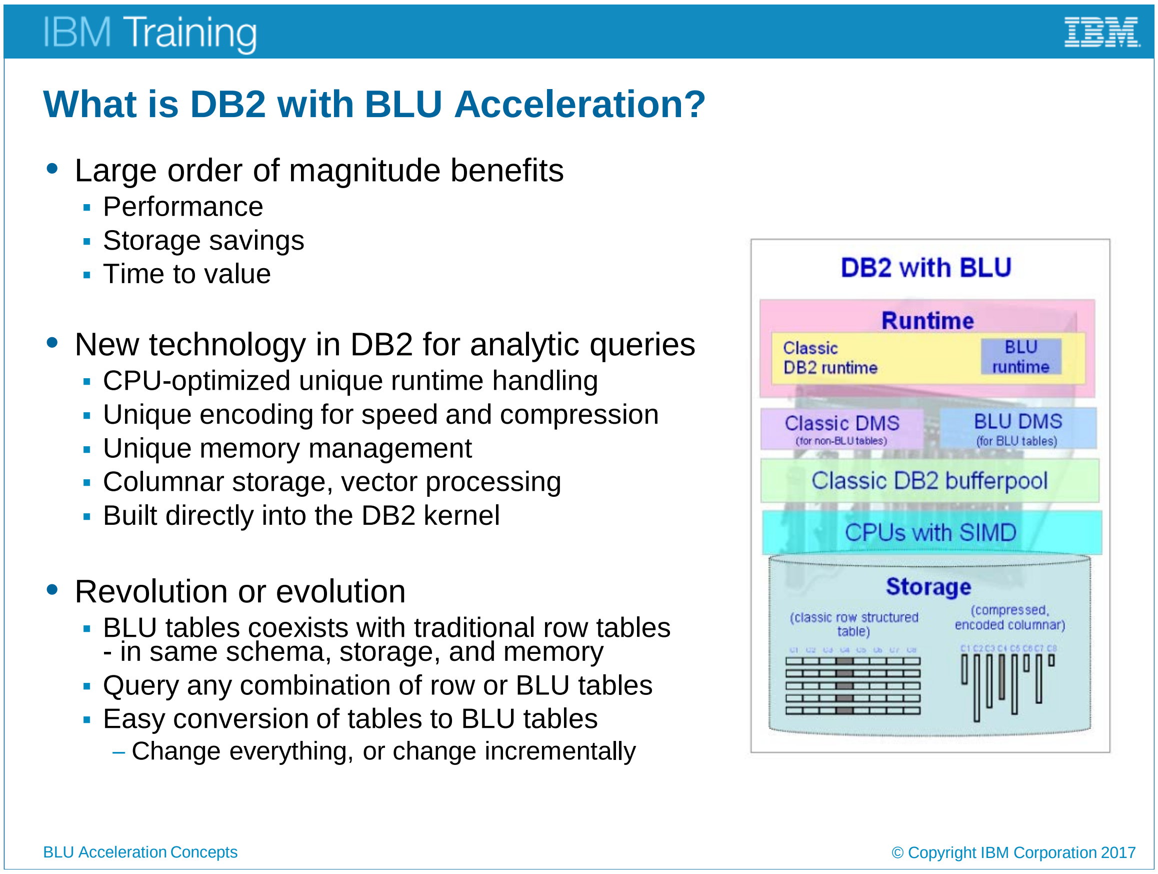
Unit objectives

- List the seven ‘big ideas’ that work together to provide DB2 BLU Acceleration
- Describe how the column dictionaries used to provide extreme compression of column-organized tables are built and utilized
- Explain the impact of setting the DB2 registry variable DB2_WORKLOAD to ANALYTICS
- Describe the different storage objects used for column-organized tables compared to row-organized tables, including the special page map index.
- Explain how DB2 uses a synopsis table to support data skipping with DB2 BLU Acceleration, column-organized tables

BLU Acceleration Concepts

© Copyright IBM Corporation 2017

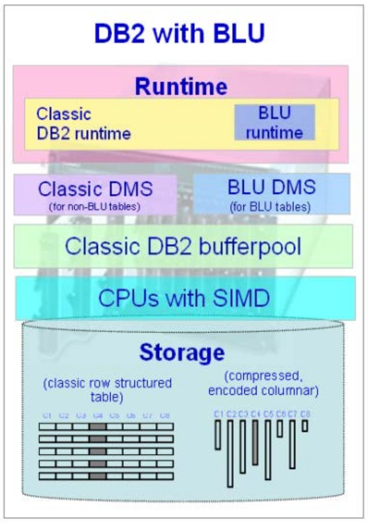
Unit Objectives



IBM Training

What is DB2 with BLU Acceleration?

- Large order of magnitude benefits
  - Performance
  - Storage savings
  - Time to value
- New technology in DB2 for analytic queries
  - CPU-optimized unique runtime handling
  - Unique encoding for speed and compression
  - Unique memory management
  - Columnar storage, vector processing
  - Built directly into the DB2 kernel
- Revolution or evolution
  - BLU tables coexists with traditional row tables - in same schema, storage, and memory
  - Query any combination of row or BLU tables
  - Easy conversion of tables to BLU tables
    - Change everything, or change incrementally



BLU Acceleration Concepts

© Copyright IBM Corporation 2017

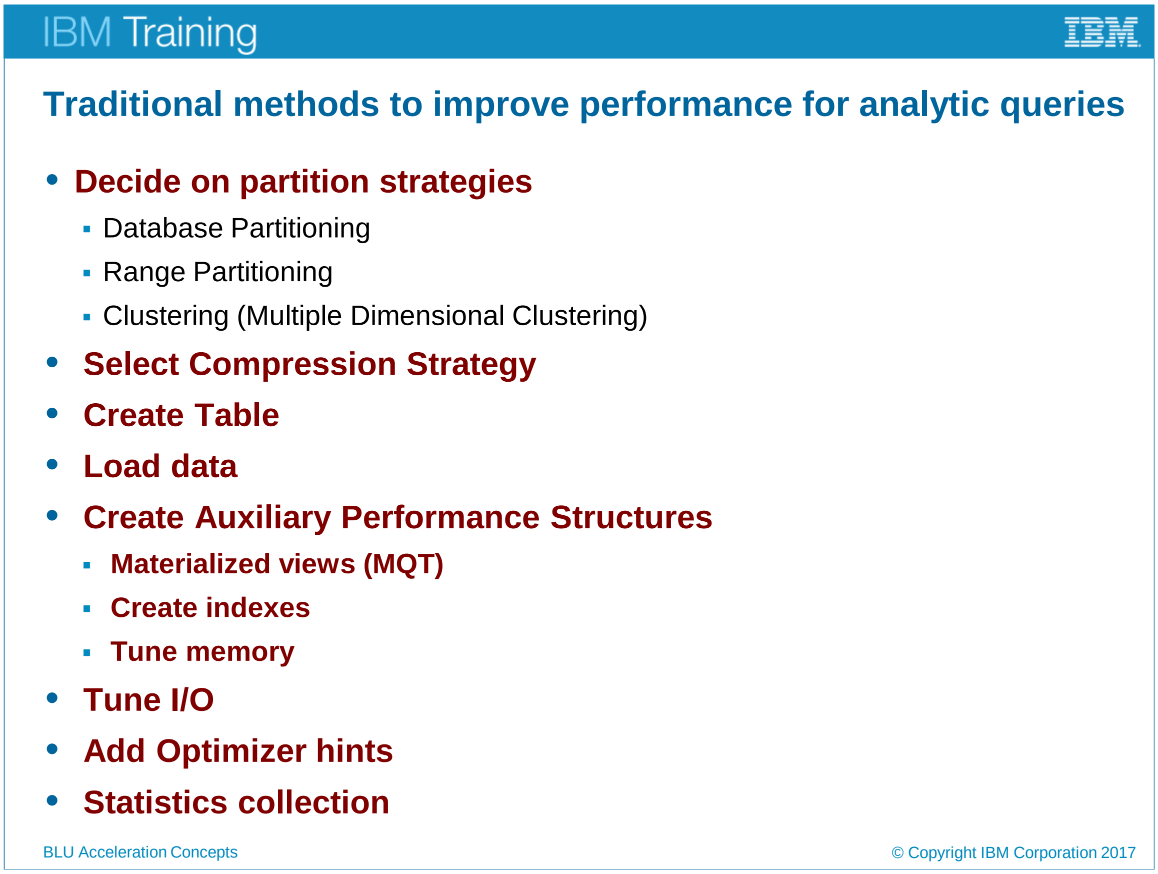
What is DB2 with BLU Acceleration?

This slide describes at a high level what DB2 with BLU Acceleration is. What is the key business value of implementing BLU Acceleration?

This is a new technology that has been developed by IBM and integrated directly into the DB2 engine. BLU Acceleration is a new storage engine along with integrated runtime (directly into the core DB2 engine) to support the storage and analysis of column-organized tables. The BLU Acceleration processing is parallel to the regular, row-based table processing found in the DB2 engine. This is not a bolt-on technology nor is it a separate analytic engine that sits outside of DB2. Much like when IBM added XML data as a first class object within the database along with all the storage and processing enhancements that came with XML, now IBM has added column-organized tables directly into the storage and processing engine of DB2.

Simply put, this is a column-organized table store in DB2. Along with this store are many benefits including significantly improved performance, massive storage savings and ease of implementation and ease of management.

This feature allows us to deliver on these performance and storage innovations while also optimizing the use of main-memory, improving I/O efficiency and exploiting CPU instructions and characteristics to enhance the value derived from your database investments.



IBM Training

Traditional methods to improve performance for analytic queries

- **Decide on partition strategies**
  - Database Partitioning
  - Range Partitioning
  - Clustering (Multiple Dimensional Clustering)
- **Select Compression Strategy**
- **Create Table**
- **Load data**
- **Create Auxiliary Performance Structures**
  - **Materialized views (MQT)**
  - **Create indexes**
  - **Tune memory**
- **Tune I/O**
- **Add Optimizer hints**
- **Statistics collection**

BLU Acceleration Concepts © Copyright IBM Corporation 2017

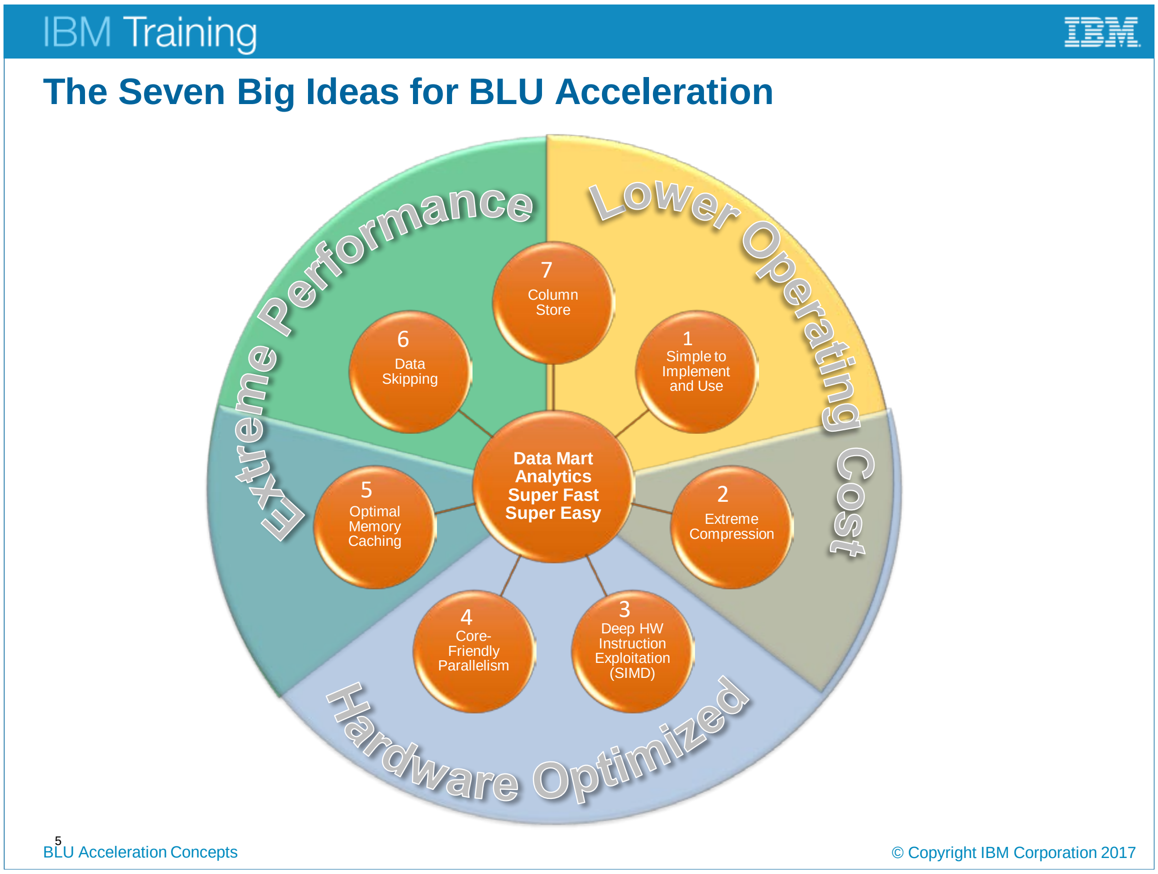
Traditional methods to improve performance for analytic queries

The slide lists some of the steps that would traditionally be used to optimize performance for relational database applications.

- Decide on partition strategies - Before a table is created in a database the data partitioning strategies that would best fit the application would be determined including:
  - Database partitioning - You might decide to use a partitioned database cluster to provide the benefits of inter-partition parallel processing.
  - Range partitioning - You might decide to implement a range partitioned table which can use partition elimination to improve performance.
  - Clustering - You may also consider clustering data or using multi-dimensional clustering depending on the type of access used for tables
- Select the compression strategy - With DB2 10.1 and later, you could choose the static classic compression or add adaptive compression for tables to reduce system I/O and memory resources for access to large tables.
- Create the table - Next the table would be defined.

- Load the data - Once the table is defined, the table data can be loaded using the DB2 LOAD utility, another DB2 utility like INGEST, or using application processing.
- Create Auxiliary Performance Structures - In order to achieve good performance for the application additional supporting database objects might be created, like indexes or materialized query tables (MQT). You may also adjust buffer pool memory allocations to improve access performance for a table. These may require additional tuning over time as table size changes and new application queries are developed.
- Tuning I/O - In some cases you may decide to tune the I/O access resources for a table. For example, you might move a table to a new high performance device, like solid state storage or implement multi-temperature storage for a range partitioned table.
- Add Optimizer hints - For a query that does not perform well using the standard access plan generated by the SQL compiler, you might provide optimizer hints, in DB2 LUW this is done using optimization profiles.
- Statistics Collection - You might find that the default table statistics are not sufficient to generate efficient access plans for some queries, so special detailed table and index statistics might be collected. With DB2 LUW you can define statistical views and collect statistics on those to supplement the table statistics.

We will see that with column-organized tables, we can reduce the task list to two steps, create the table and then load the data.



IBM Training

The Seven Big Ideas for BLU Acceleration

Extreme Performance
Lower Operating Cost
Hardware Optimized
Data Mart Analytics Super Fast Super Easy
7 Column Store
1 Simple to Implement and Use
2 Extreme Compression
3 Deep HW Instruction Exploitation (SIMD)
4 Core-Friendly Parallelism
5 Optimal Memory Caching
6 Data Skipping

5 BLU Acceleration Concepts © Copyright IBM Corporation 2017

The Seven Big Ideas for BLU Acceleration

BLU Acceleration is based on seven innovations that have been added to DB2 which we call “Big Ideas”. Each of these big ideas is a technology capability that provides business value and together, make up what we refer to as **BLU Acceleration.**

Lower Operation Costs

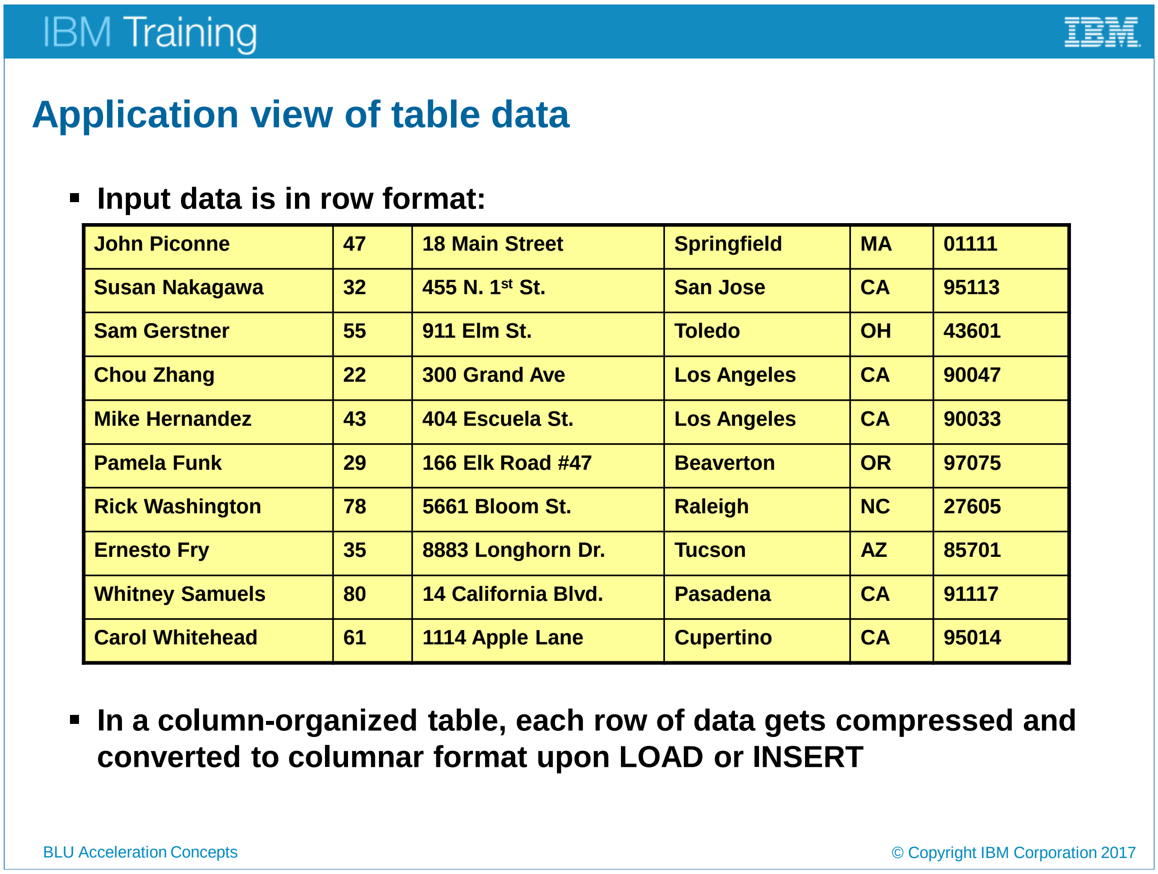
- Column Store – by storing table data in column-organized format we not only save significantly on storage costs but we also improve I/O and memory efficiency. This lowers operating costs.
- Simple to Implement and Use – with BLU Acceleration you just create the column-organized table and then load and go. The additional steps like memory tuning, selection and creation of indexes, etc. are eliminated. This lowers administration and development costs significantly
- Extreme Compression – by using compression and sophisticated encoding algorithms, DB2 can save significantly on storage costs including power, cooling, and management of that storage.

Hardware Optimized

- Extreme Compression – in addition to the lower costs, the compression algorithms used exploit processor characteristics to improve performance. The compression we use works with a register friendly encoding technique to improve processor efficiency
- Deep Hardware Instruction Exploitation – we will discuss this in more detail later in this lecture, but with SIMD processing we are multiplying the performance of the processor by having instructions work on multiple data elements simultaneously.
- Core Friendly Parallelism – Access plans on column-organized tables will leverage all of the cores on the server simultaneously to deliver better analytic query performance
- Optimal Memory Caching – With row-organized tables, a full table scan ends up putting data into the bufferpool that is often not required. For column-organized tables, if there are columns that are involved in joins or other predicates in many queries then we can pack the bufferpool full of those columns while keeping other columns out of memory if they are not regularly used. This improves performance and optimizes the memory available

Extreme Performance

- Optimal Memory Caching – as stated above, this not only helps to optimize hardware but also improves overall workload performance
- Data Skipping – by keeping track of which pages of data contain which column values, we can reduce the I/O costs for query processing by simply skipping data we already know would not qualify for the query.
- Column Store – in addition to lowering costs, by selecting only columns that are part of a query we can increase performance of queries by an order of magnitude in some cases.



IBM Training

Application view of table data

- **Input data is in row format:**

John Piconne	47	18 Main Street	Springfield	MA	01111
Susan Nakagawa	32	455 N. 1st 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

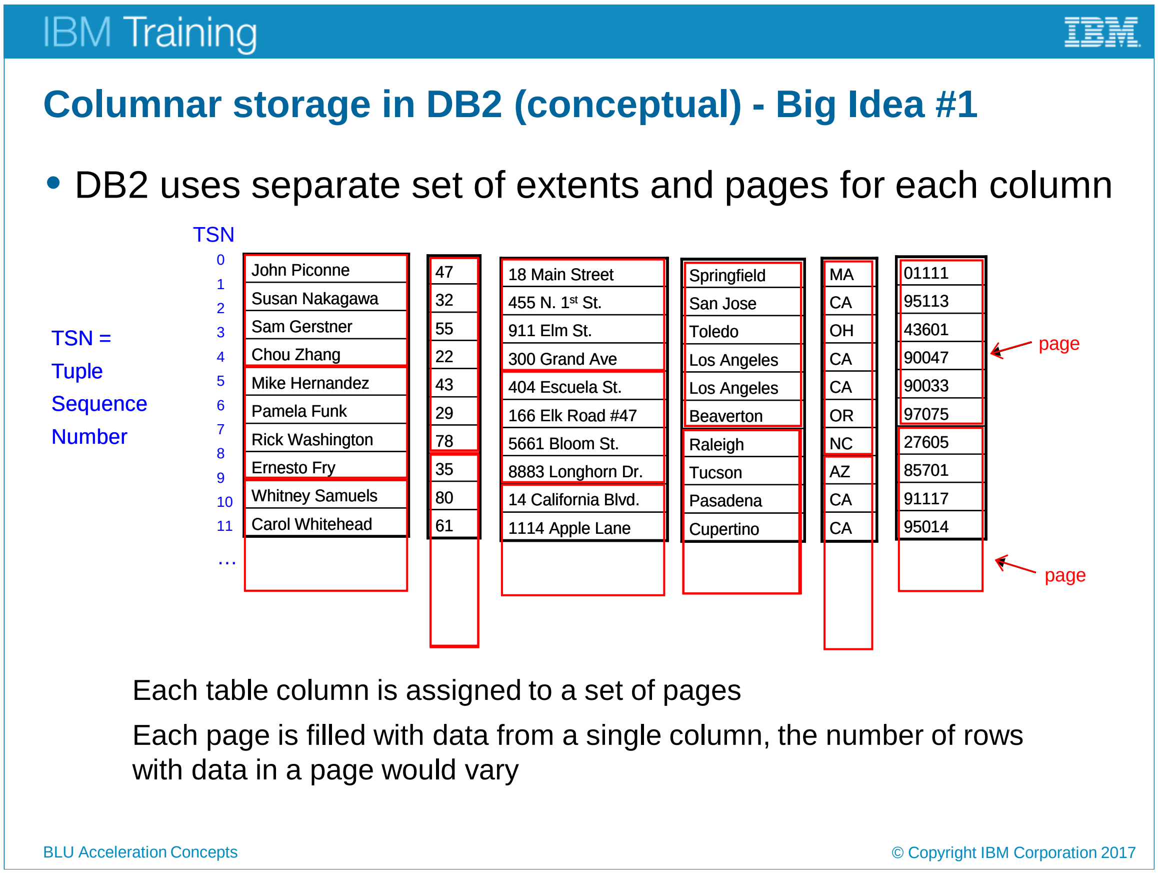
- **In a column-organized table, each row of data gets compressed and converted to columnar format upon LOAD or INSERT**

BLU Acceleration Concepts © Copyright IBM Corporation 2017

Application view of table data

The standard application view of the data is that of a relational table with rows and columns does not change when using column-organized tables.

The column-organized tables can be loaded using the DB2 LOAD utility from the same row oriented input, or applications can use SQL INSERT statements to populate the tables. As new rows of data are stored, DB2 will compress the data and convert it to the columnar format.



IBM Training

Columnar storage in DB2 (conceptual) - Big Idea #1

- DB2 uses separate set of extents and pages for each column

TSN						
0	John Piconne	47	18 Main Street	Springfield	MA	01111
1	Susan Nakagawa	32	455 N. 1st 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						
…						

Each table column is assigned to a set of pages

Each page is filled with data from a single column, the number of rows with data in a page would vary

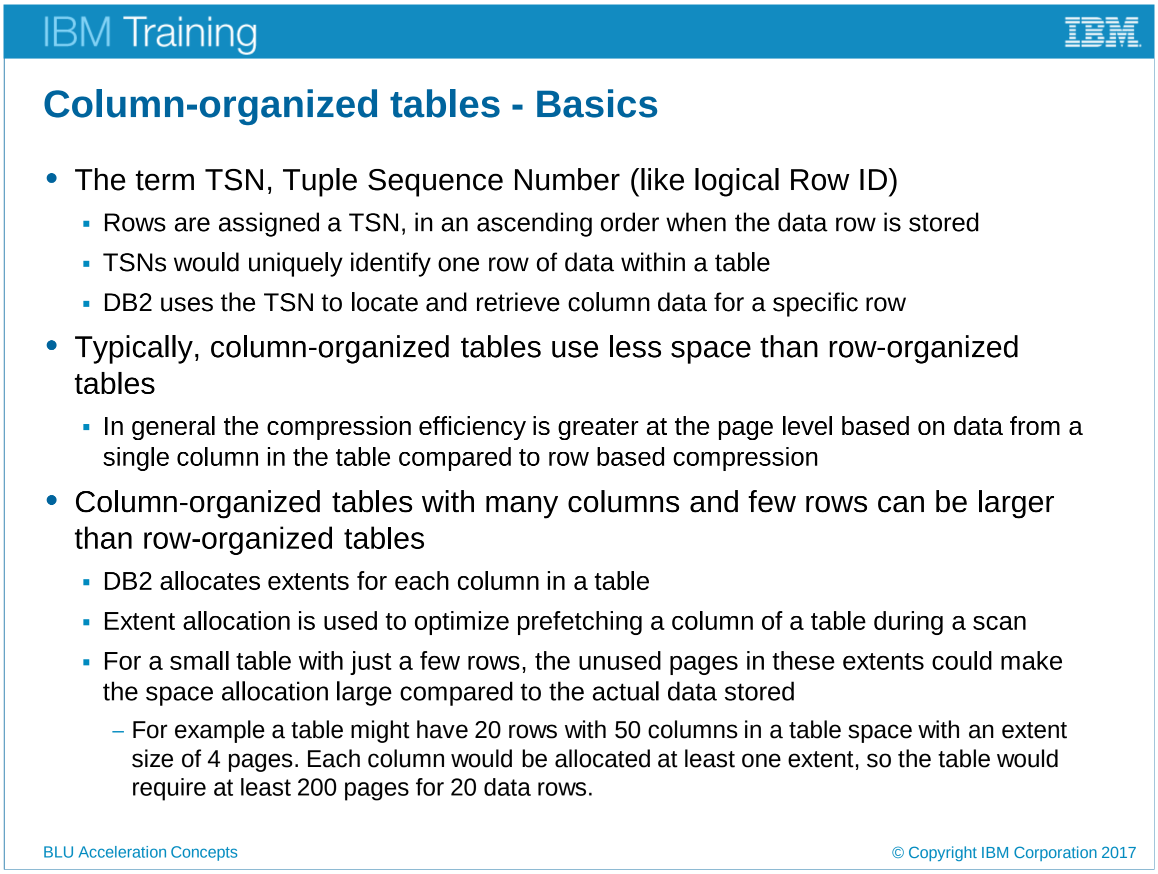
BLU Acceleration Concepts © Copyright IBM Corporation 2017

Columnar storage in DB2 (conceptual) - Big Idea #1

DB2 allocates extents of pages for each column in a column-organized table. The page size and extent size is fixed for each table, based on the table space assigned when the CREATE TABLE statement is executed.

Each page will only contain data from a single column in the table. The number of rows that share a page will vary.

The visual shows the concept of storing data columns on different pages. We will see that compression techniques are used for every column of data. The visual shows uncompressed data for ease of understanding.



IBM Training

Column-organized tables - Basics

- The term TSN, Tuple Sequence Number (like logical Row ID)
  - Rows are assigned a TSN, in an ascending order when the data row is stored
  - TSNs would uniquely identify one row of data within a table
  - DB2 uses the TSN to locate and retrieve column data for a specific row
- Typically, column-organized tables use less space than row-organized tables
  - In general the compression efficiency is greater at the page level based on data from a single column in the table compared to row based compression
- Column-organized tables with many columns and few rows can be larger than row-organized tables
  - DB2 allocates extents for each column in a table
  - Extent allocation is used to optimize prefetching a column of a table during a scan
  - For a small table with just a few rows, the unused pages in these extents could make the space allocation large compared to the actual data stored
    - For example a table might have 20 rows with 50 columns in a table space with an extent size of 4 pages. Each column would be allocated at least one extent, so the table would require at least 200 pages for 20 data rows.

BLU Acceleration Concepts © Copyright IBM Corporation 2017

Column-organized tables - Basics

DB2 uses a tuple sequence number (TSN) to manage rows of data stored in column-organized tables. The TSN is similar to a unique row identifier and is used internally by DB2, applications so not need to reference this value.

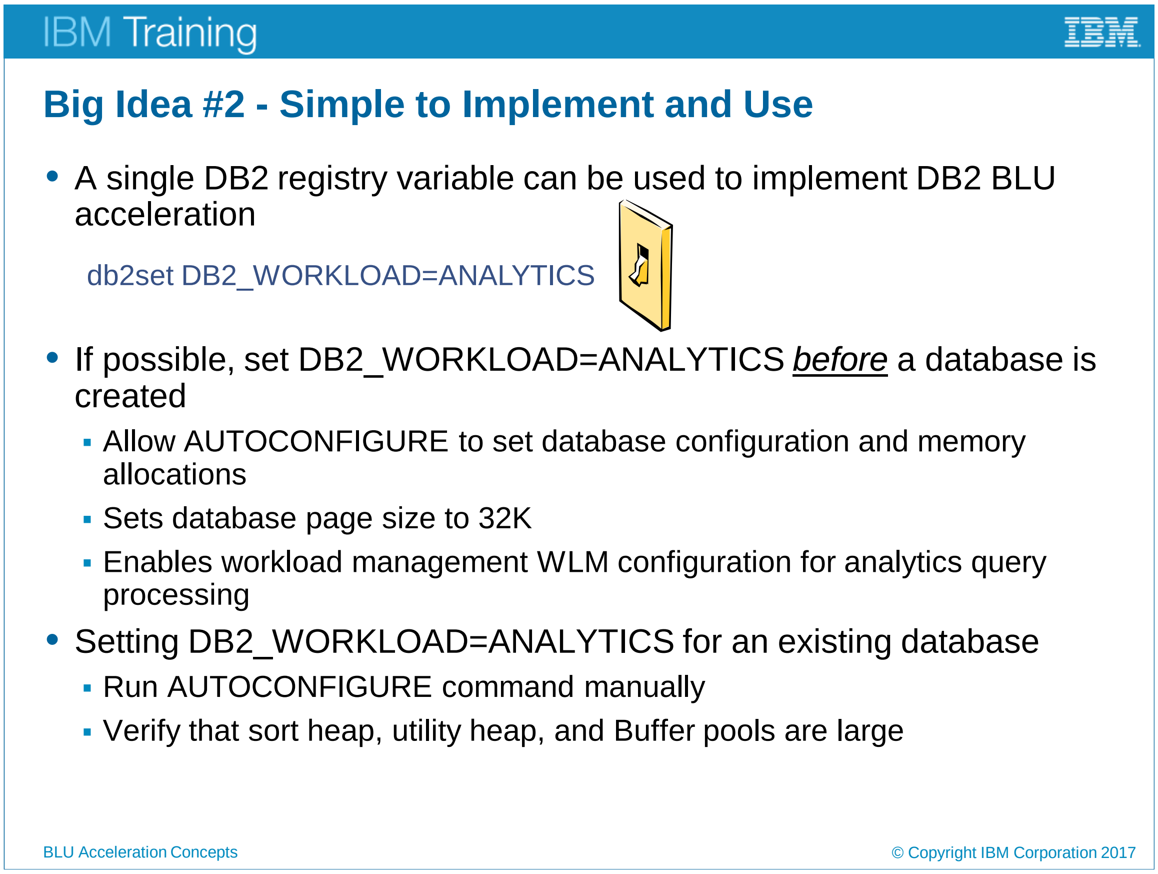
The TSN for a row of data is unique within a table. It is used by DB2 to pull together all of the column data for a row that is being selected by an application.

In general, the compression results for a column-organized table exceeds the compression results for a similar row-organized table. One exception to this concept is for tables that have a small number of data rows. In order to efficiently scan data for one column of a column-organized table, DB2 allocates unique extents for each data column.

The visual describes the example of a small table with the following characteristics:

- The table has just 20 rows of data
- The table was created with 50 columns
- The table space used to create the table was defined with an extent size of four pages

Since each of the 50 columns would require at least one extent of four pages, the table would require at least 200 pages of storage. This is an unusual case where column organization likely requires more storage than row organization for a table.



IBM Training

Big Idea #2 - Simple to Implement and Use

- A single DB2 registry variable can be used to implement DB2 BLU acceleration

  db2set DB2_WORKLOAD=ANALYTICS

- If possible, set DB2_WORKLOAD=ANALYTICS *before* a database is created
  - Allow AUTOCONFIGURE to set database configuration and memory allocations
  - Sets database page size to 32K
  - Enables workload management WLM configuration for analytics query processing
- Setting DB2_WORKLOAD=ANALYTICS for an existing database
  - Run AUTOCONFIGURE command manually
  - Verify that sort heap, utility heap, and Buffer pools are large

BLU Acceleration Concepts © Copyright IBM Corporation 2017

Big Idea #2 - Simple to Implement and Use

DB2 column-organized tables add columnar capabilities to DB2 databases, which includes data stored with column organization and vector processing of column data. Using this table format with star schema data marts provides significant improvements to storage, query performance, and ease of use through simplified design and tuning.

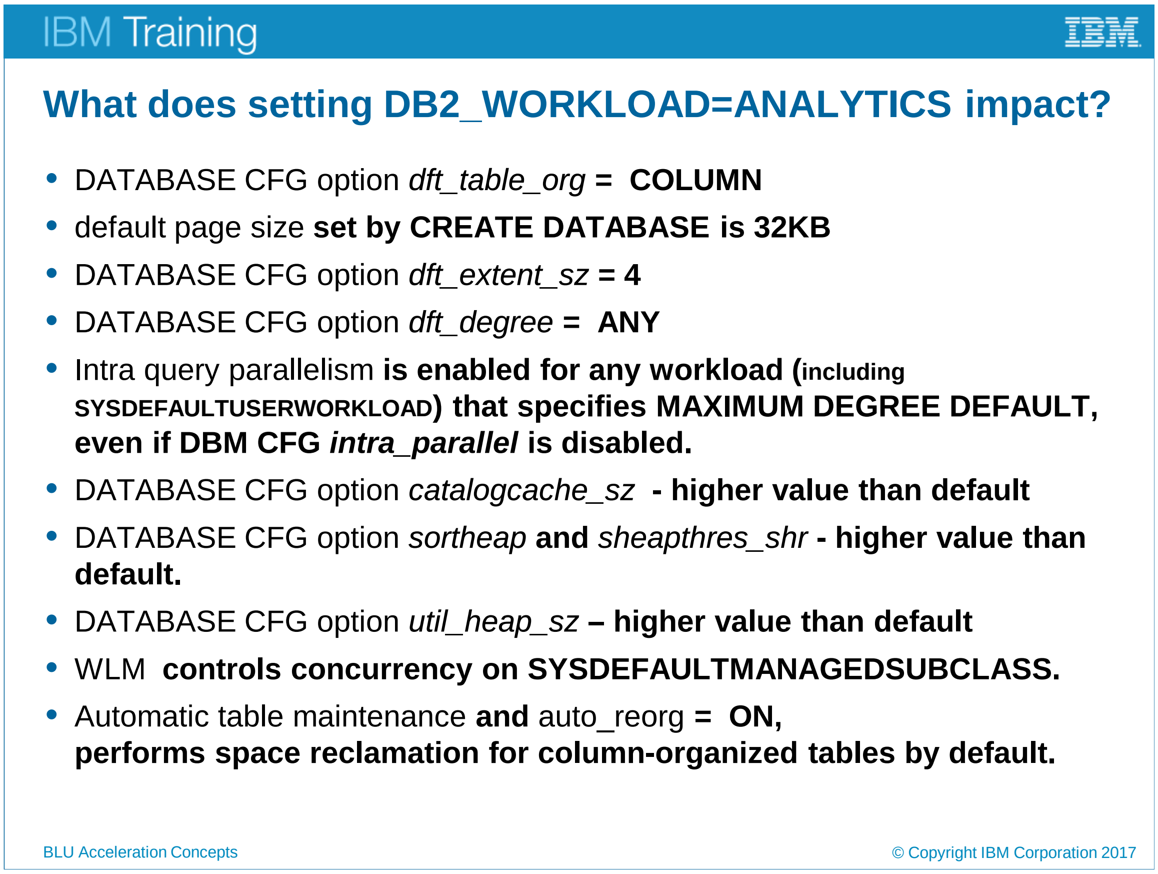
If the majority of tables in your database will be column-organized tables, set the **DB2_WORLOAD** registry variable to **ANALYTICS** prior to creating the database. Doing so helps to configure memory, table organization, page size, and extent size, and enables workload management.

The recommended approach is to put as many tables into column-organized format as possible, if the workload is entirely an analytics/OLAP workload.

These workloads are characterized by non-selective data access (that is, queries access more than approximately 5% of the data), and extensive scanning, grouping, and aggregation.

Workloads that are transactional in nature should not use column-organized tables. Traditional row-organized tables with index access are generally better suited for these environments.

In the case of mixed workloads, which include a combination of analytic query processing and very selective access (involving less than 2% of the data), a mix of row-organized and column-organized tables might be suitable.



IBM Training

What does setting DB2_WORKLOAD=ANALYTICS impact?

- DATABASE CFG option *dft_table_org* **= COLUMN**
- default page size **set by CREATE DATABASE is 32KB**
- DATABASE CFG option *dft_extent_sz* **= 4**
- DATABASE CFG option *dft_degree* **= ANY**
- Intra query parallelism **is enabled for any workload (including SYSDEFAULTUSERWORKLOAD) that specifies MAXIMUM DEGREE DEFAULT, even if DBM CFG *intra_parallel* is disabled.**
- DATABASE CFG option *catalogcache_sz* **- higher value than default**
- DATABASE CFG option *sortheap* **and** *sheapthres_shr* **- higher value than default.**
- DATABASE CFG option *util_heap_sz* **– higher value than default**
- WLM **controls concurrency on SYSDEFAULTMANAGEDSUBCLASS.**
- Automatic table maintenance **and** auto_reorg **= ON, performs space reclamation for column-organized tables by default.**

BLU Acceleration Concepts © Copyright IBM Corporation 2017

What does setting DB2_WORKLOAD=ANALYTICS impact?

Setting the DB2 registry variable DB2_WORKLOAD to ANALYTICS prior to creating the database) will establish an optimal default configuration when using the database for analytic workloads.

The ANALYTICS option ensures that the following configuration settings are performed automatically (unless AUTOCONFIGURE is disabled):

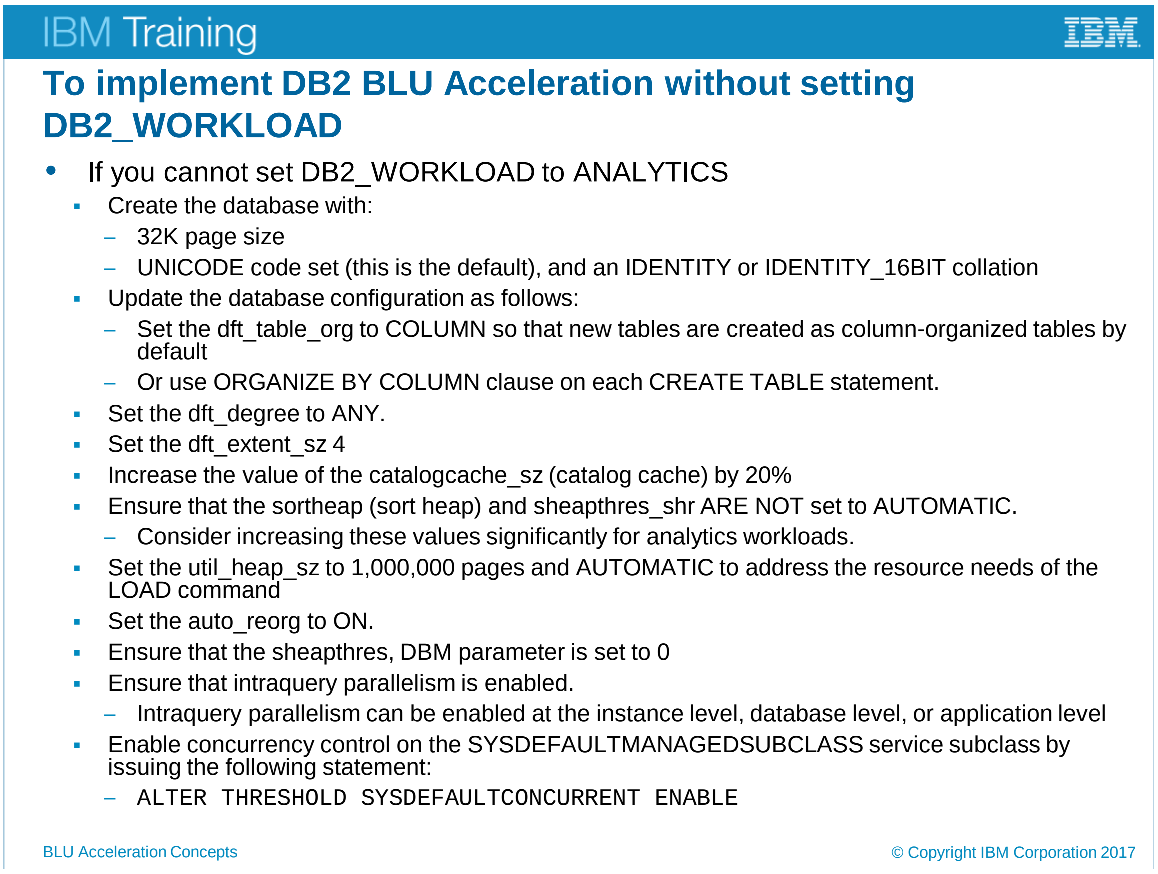
- The **dft_table_org** (default table organization for user tables) database configuration parameter is set to COLUMN.
- The **dft_degree** (default degree) database configuration parameter is set to ANY.
- The **dft_extent_sz** (default extent size) database configuration parameter is set to 4.
- The **catalogcache_sz** (catalog cache) database configuration parameter is set to a higher value than that for a non-analytics workload.
- The **sortheap** (sort heap) and **sheapthres_shr** (sort heap threshold for shared sorts) database configuration parameters are calculated specifically for an analytics workload, and take into account the additional memory requirements for processing column-organized data.

- The **util_heap_sz** (utility heap size) database configuration parameter is set to a value that takes into account the additional memory that is required to load the data into column-organized tables.
- The **auto_reorg** (automatic reorganization) database configuration parameter is set to ON.

Running the **AUTOCONFIGURE** command against an existing database when DB2_WORKLOAD is set to ANALYTICS has the same result.

The following additional choices are made automatically:

- The default database page size for a newly created database is set to 32K.
- A larger database shared sort heap is allocated.
- Intra-query parallelism is enabled for any workload (including SYSDEFAULTUSERWORKLOAD) that specifies MAXIMUM DEGREE DEFAULT, even if intra_parallel is disabled.
- Concurrency control is enabled on SYSDEFAULTMANAGEDSUBCLASS.
- Automatic table maintenance performs space reclamation for column-organized tables by default.



IBM Training

To implement DB2 BLU Acceleration without setting DB2_WORKLOAD

- If you cannot set DB2_WORKLOAD to ANALYTICS
  - Create the database with:
    - 32K page size
    - UNICODE code set (this is the default), and an IDENTITY or IDENTITY_16BIT collation
  - Update the database configuration as follows:
    - Set the dft_table_org to COLUMN so that new tables are created as column-organized tables by default
    - Or use ORGANIZE BY COLUMN clause on each CREATE TABLE statement.
  - Set the dft_degree to ANY.
  - Set the dft_extent_sz 4
  - Increase the value of the catalogcache_sz (catalog cache) by 20%
  - Ensure that the sortheap (sort heap) and sheapthres_shr ARE NOT set to AUTOMATIC.
    - Consider increasing these values significantly for analytics workloads.
  - Set the util_heap_sz to 1,000,000 pages and AUTOMATIC to address the resource needs of the LOAD command
  - Set the auto_reorg to ON.
  - Ensure that the sheapthres, DBM parameter is set to 0
  - Ensure that intraquery parallelism is enabled.
    - Intraquery parallelism can be enabled at the instance level, database level, or application level
  - Enable concurrency control on the SYSDEFAULTMANAGEDSUBCLASS service subclass by issuing the following statement:
    - `ALTER THRESHOLD SYSDEFAULTCONCURRENT ENABLE`

BLU Acceleration Concepts © Copyright IBM Corporation 2017

To implement DB2 BLU Acceleration without setting DB2_WORKLOAD

If you cannot create your database and have it auto-configured while DB2_WORKLOAD=ANALYTICS, take the following steps to create and optimally configure your database for analytic workloads.

1. Create the database with a 32K page size, a UNICODE code set (this is the default), and an IDENTITY or IDENTITY_16BIT collation.
   For example:

```
CREATE DATABASE DMART COLLATE USING IDENTITY PAGESIZE 32 K
```

2. Update the database configuration as follows:

   - Set the dft_table_org (default table organization for user tables) database configuration parameter to COLUMN so that new tables are created as column-organized tables by default; otherwise, the ORGANIZE BY COLUMN clause must be specified on each CREATE TABLE statement.

   - Set the dft_degree (default degree) database configuration parameter to ANY.

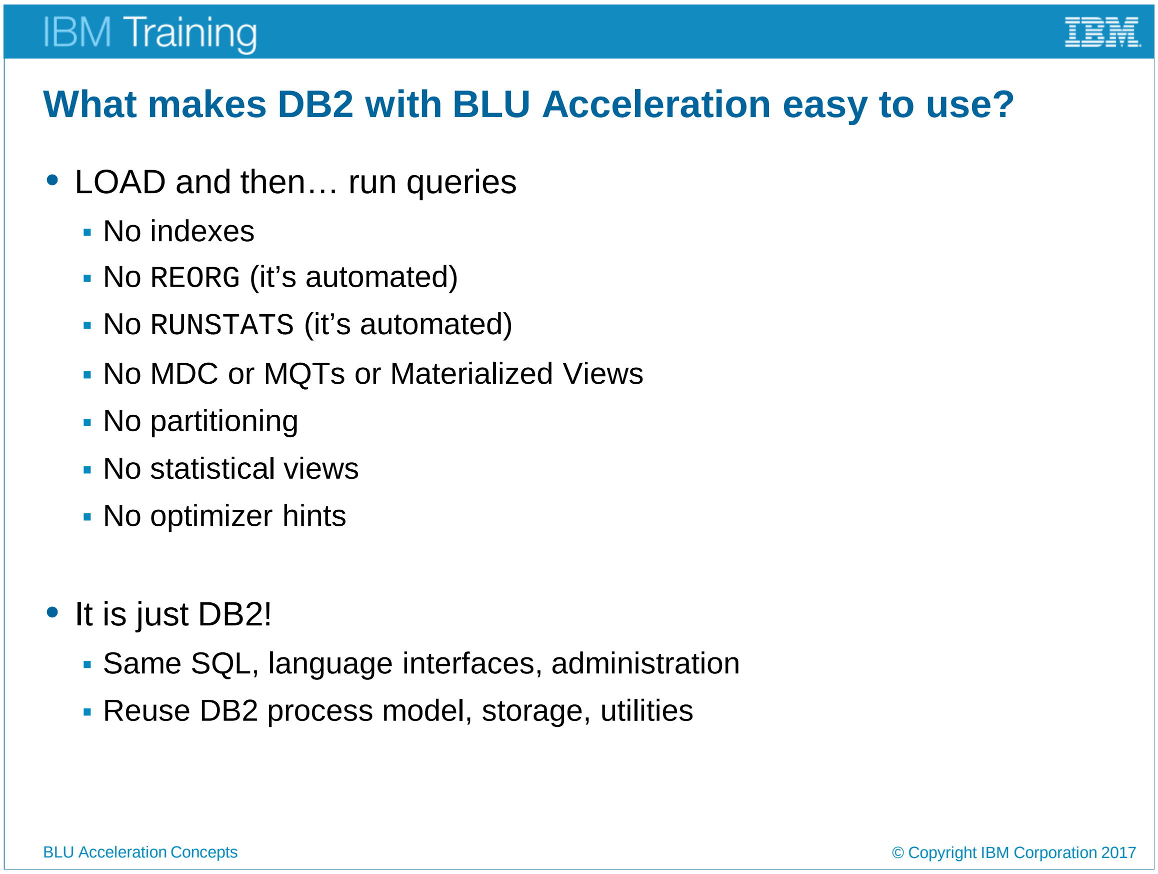
   - Set the dft_extent_sz (default extent size) database configuration parameter to 4.

- Increase the value of the catalogcache_sz (catalog cache) database configuration parameter by 20% (it is set automatically during database creation).
- Ensure that the sortheap (sort heap) and sheapthres_shr (sort heap threshold for shared sorts) database configuration parameters are not set to AUTOMATIC. Consider increasing these values significantly for analytics workloads. A reasonable starting point is setting sheapthres_shr to the size of the buffer pool (across all buffer pools). Set sortheap to some fraction (for example, 1/20) of sheapthres_shr to enable concurrent sort operations.
- Set the util_heap_sz (utility heap size) database configuration parameter to 1,000,000 pages and AUTOMATIC to address the resource needs of the LOAD command. If the database server has at least 128 GB of memory, set util_heap_sz to 4,000,000 pages. If concurrent load operations are running, increase the value of util_heap_sz to accommodate higher memory requirements.
- Set the auto_reorg (automatic reorganization) database configuration parameter to ON.

These changes will increase the overall database memory that is required for your database. Consider increasing the database_memory configuration parameter for the database if this parameter was not already set to AUTOMATIC.

3. Ensure that the sheapthres database manager configuration parameter is set to 0 (this is the default value). Note that this setting applies to all databases in the instance.
4. Ensure that intraquery parallelism, which is required to access column-organized tables, is enabled. Intraquery parallelism can be enabled at the instance level, database level, or application level; for details, see intraquery parallelism and intrapartition parallelism.
5. Enable concurrency control on the SYSDEFAULTMANAGEDSUBCLASS service subclass by issuing the following statement:

```
ALTER THRESHOLD SYSDEFAULTCONCURRENT ENABLE
```



IBM Training

What makes DB2 with BLU Acceleration easy to use?

- LOAD and then… run queries
  - No indexes
  - No `REORG` (it's automated)
  - No `RUNSTATS` (it's automated)
  - No MDC or MQTs or Materialized Views
  - No partitioning
  - No statistical views
  - No optimizer hints
- It is just DB2!
  - Same SQL, language interfaces, administration
  - Reuse DB2 process model, storage, utilities

BLU Acceleration Concepts © Copyright IBM Corporation 2017

What makes DB2 with BLU Acceleration easy to use?

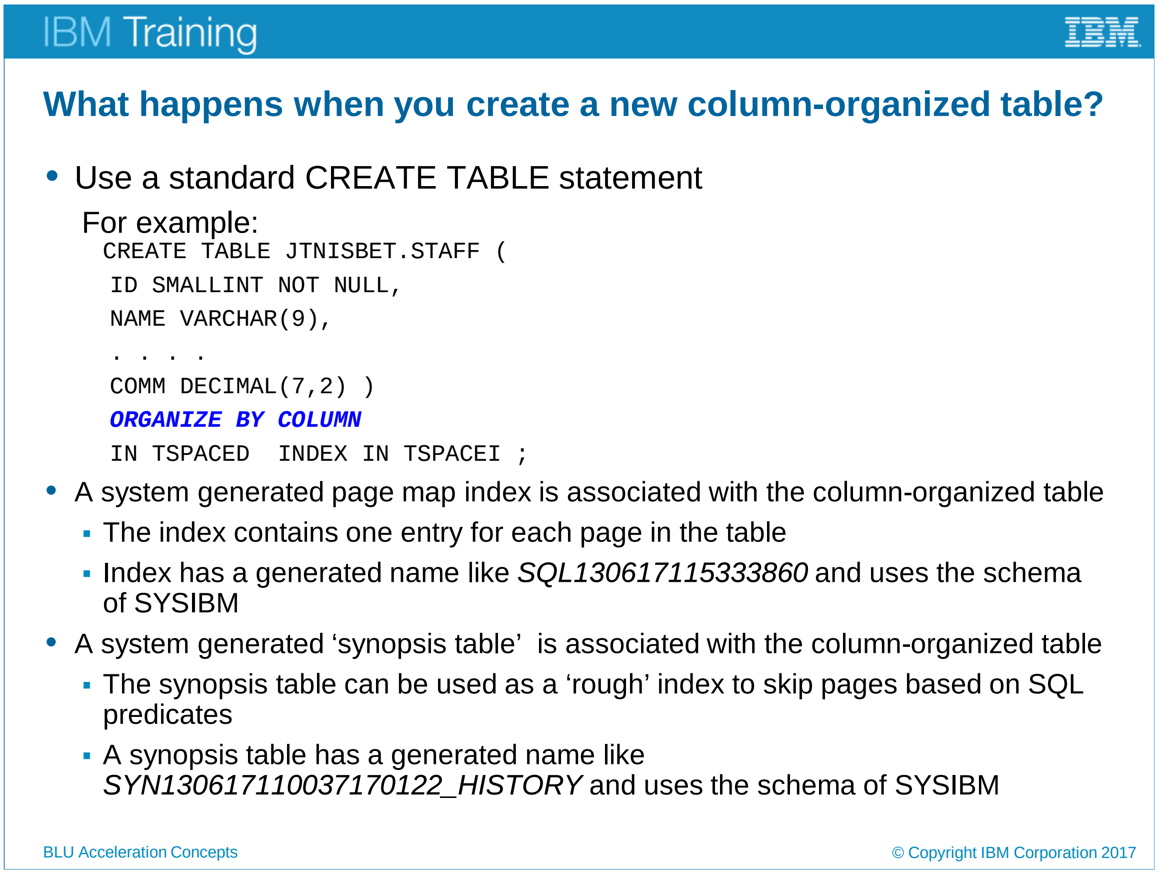
The simplicity of DB2 with BLU Acceleration is one of the key value propositions. The fact that it is really simple to use is key. Remember it is all part of the DB2 kernel.

Because of the simplicity and the built in space reclamation and statistics gathering, and also the fact that other structures such as indexes, MDC tables, MQT tables, etc are not needed really to add to the value.

From a customer perspective it is LOAD and GO! You can start running your queries immediately after data load and start getting the performance gains of DB2 with BLU Acceleration immediately.

Setting DB2_WORKLOAD to ANALYTICS does all the work:

- Automatically configures DB2 for optimal analytics performance
- Makes column-organized tables the default table type
- Enables automatic workload management
- Enables automatic space reclaim
- Page and extent size configured for analytics
- Memory for caching, sorting and hashing, utilities are automatically initialized based on the server size and available RAM



IBM Training

What happens when you create a new column-organized table?

- Use a standard CREATE TABLE statement

  For example:

```
CREATE TABLE JTNISBET.STAFF (
 ID SMALLINT NOT NULL,
 NAME VARCHAR(9),
 . . . .
 COMM DECIMAL(7,2) )
 ORGANIZE BY COLUMN
 IN TSPACED  INDEX IN TSPACEI ;
```

- A system generated page map index is associated with the column-organized table
  - The index contains one entry for each page in the table
  - Index has a generated name like *SQL130617115333860* and uses the schema of SYSIBM
- A system generated ‘synopsis table’ is associated with the column-organized table
  - The synopsis table can be used as a ‘rough’ index to skip pages based on SQL predicates
  - A synopsis table has a generated name like *SYN130617110037170122_HISTORY* and uses the schema of SYSIBM

BLU Acceleration Concepts © Copyright IBM Corporation 2017

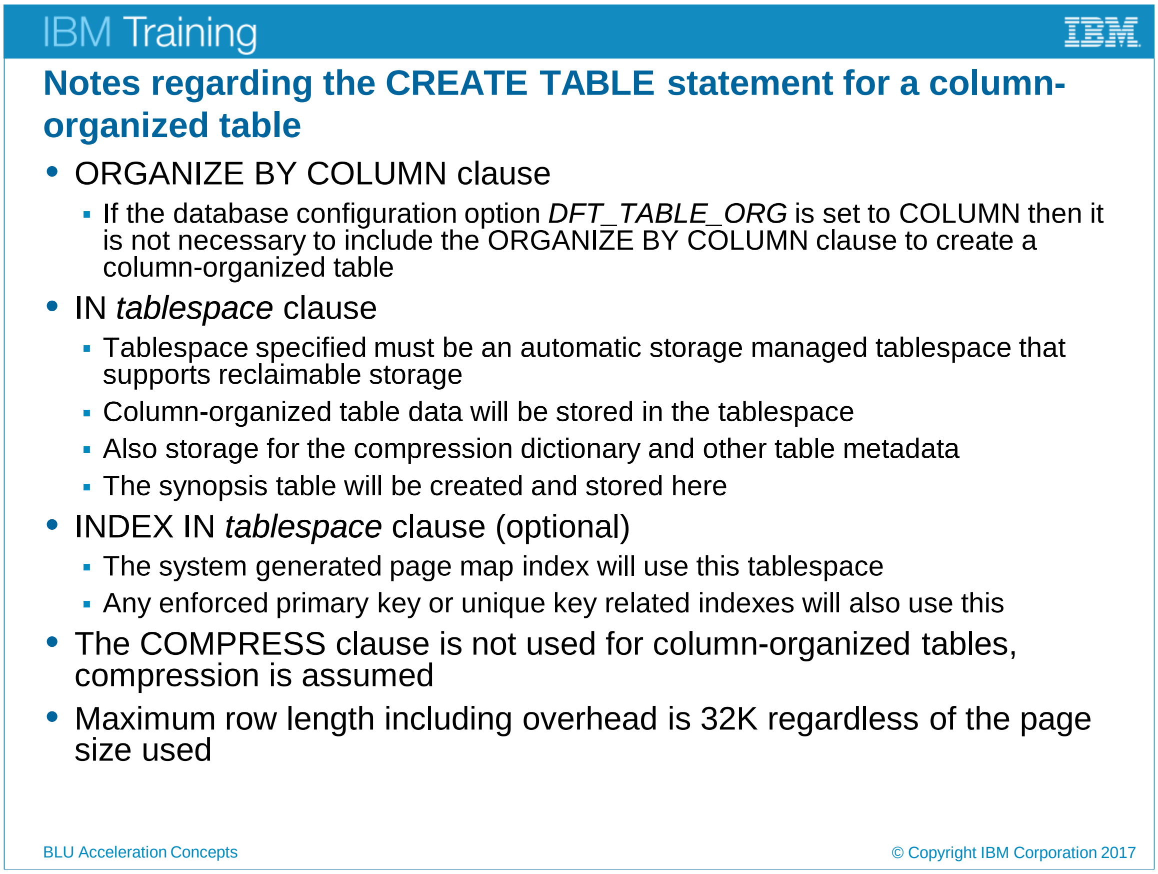
What happens when you create a new column-organized table?

To create a column-organized table, specify the ORGANIZE BY COLUMN clause on the CREATE TABLE statement.

When a column-organized table is created, DB2 creates a system generated page map index. The index contains one entry for each page in the column-organized table. The index is assigned a system generated name and uses a schema of SYSIBM.

Column organized tables can also have a system generated ‘synopsis table’. The data in the table is used as a rough index, to skip reading pages when DB2 can determine that none of the column data in the page will match an SQL predicate.

The synopsis table has a system generated name that is prefixed by ‘SYN’, and uses the schema SYSIBM.



IBM Training

Notes regarding the CREATE TABLE statement for a column-organized table

- ORGANIZE BY COLUMN clause
  - If the database configuration option *DFT_TABLE_ORG* is set to COLUMN then it is not necessary to include the ORGANIZE BY COLUMN clause to create a column-organized table
- IN *tablespace* clause
  - Tablespace specified must be an automatic storage managed tablespace that supports reclaimable storage
  - Column-organized table data will be stored in the tablespace
  - Also storage for the compression dictionary and other table metadata
  - The synopsis table will be created and stored here
- INDEX IN *tablespace* clause (optional)
  - The system generated page map index will use this tablespace
  - Any enforced primary key or unique key related indexes will also use this
- The COMPRESS clause is not used for column-organized tables, compression is assumed
- Maximum row length including overhead is 32K regardless of the page size used

BLU Acceleration Concepts © Copyright IBM Corporation 2017

Notes regarding the CREATE TABLE statement for a column-organized table

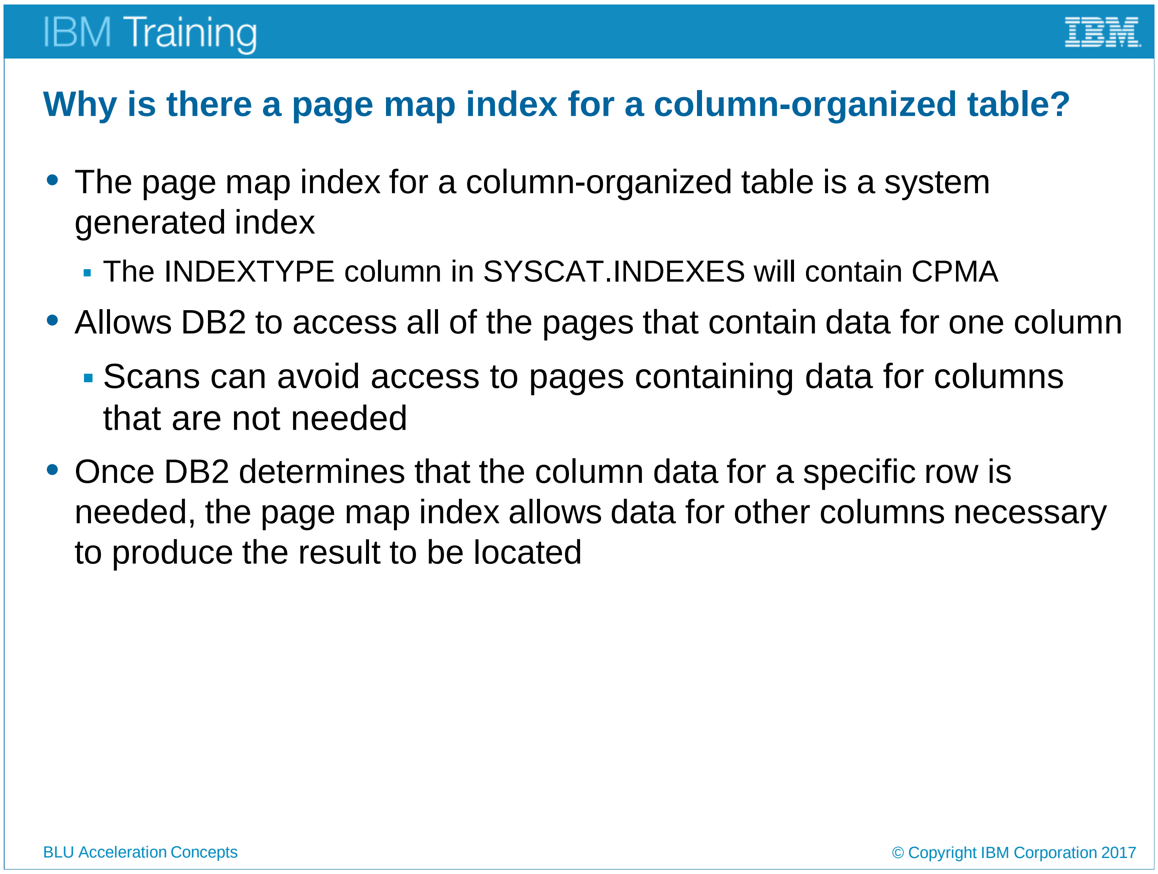
To create a column-organized table, specify the ORGANIZE BY COLUMN clause on the CREATE TABLE statement.

If you want to create tables with a specific table organization without having to specify the ORGANIZE BY COLUMN or the ORGANIZE BY ROW clause, you can change the default table organization by setting the DFT_TABLE_ORG database configuration parameter. Alternatively, you can change the default table organization to COLUMN automatically by setting the **DB2_WORKLOAD** registry variable to ANALYTICS. This setting establishes a configuration that is optimal for analytic workloads.

A column-organized table can have a maximum of 1012 columns, regardless of page size, where the byte counts of the columns must not be greater than 32,677. Extended row size support does not apply to column-organized tables.

Create column-organized tables in automatic storage table spaces only. The INDEX IN tablespace clause can be used to designate an alternate table space for storage of the page map index as well as any primary key or unique indexes defined on the table.

The COMPRESS clause is not used for column-organized table. All column-organized tables are compressed and an error will be generated if the COMPRESS option is specified when a column-organized table is created.


IBM Training

Why is there a page map index for a column-organized table?

- The page map index for a column-organized table is a system generated index
  - The INDEXTYPE column in SYSCAT.INDEXES will contain CPMA
- Allows DB2 to access all of the pages that contain data for one column
  - Scans can avoid access to pages containing data for columns that are not needed
- Once DB2 determines that the column data for a specific row is needed, the page map index allows data for other columns necessary to produce the result to be located

BLU Acceleration Concepts © Copyright IBM Corporation 2017

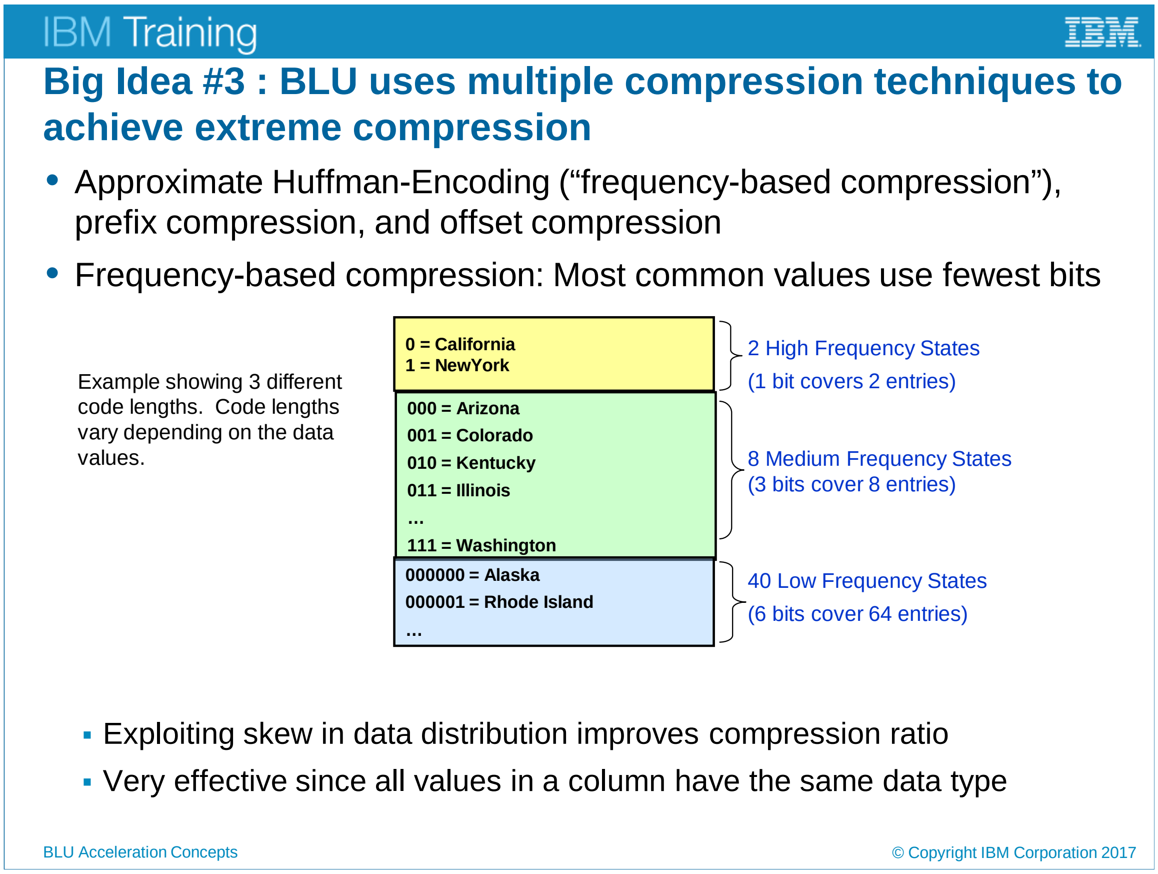
Why is there a page map index for a column-organized table?

Every column organized table has a system generated page map index. This index is used internally by DB2 to process column-organized tables. The index is based on system-generated columns, not user defined column data.

Since pages in a Column-organized table only contain data for a single table column, the page map index allows DB2 to scan a table and only read pages containing data columns referenced in a query.

The page map index allows DB2 to locate the page containing the column data for a particular row.

The index points to a page assigned to a column-organized table, not to any particular row's column value. The index size will depend on the number of pages used to store the column-organized table.



IBM Training

Big Idea #3 : BLU uses multiple compression techniques to achieve extreme compression

- Approximate Huffman-Encoding ("frequency-based compression"), prefix compression, and offset compression
- Frequency-based compression: Most common values use fewest bits

Example showing 3 different code lengths. Code lengths vary depending on the data values.

0 = California
1 = NewYork
2 High Frequency States
(1 bit covers 2 entries)
000 = Arizona
001 = Colorado
010 = Kentucky
011 = Illinois
…
111 = Washington
8 Medium Frequency States
(3 bits cover 8 entries)
000000 = Alaska
000001 = Rhode Island
…
40 Low Frequency States
(6 bits cover 64 entries)

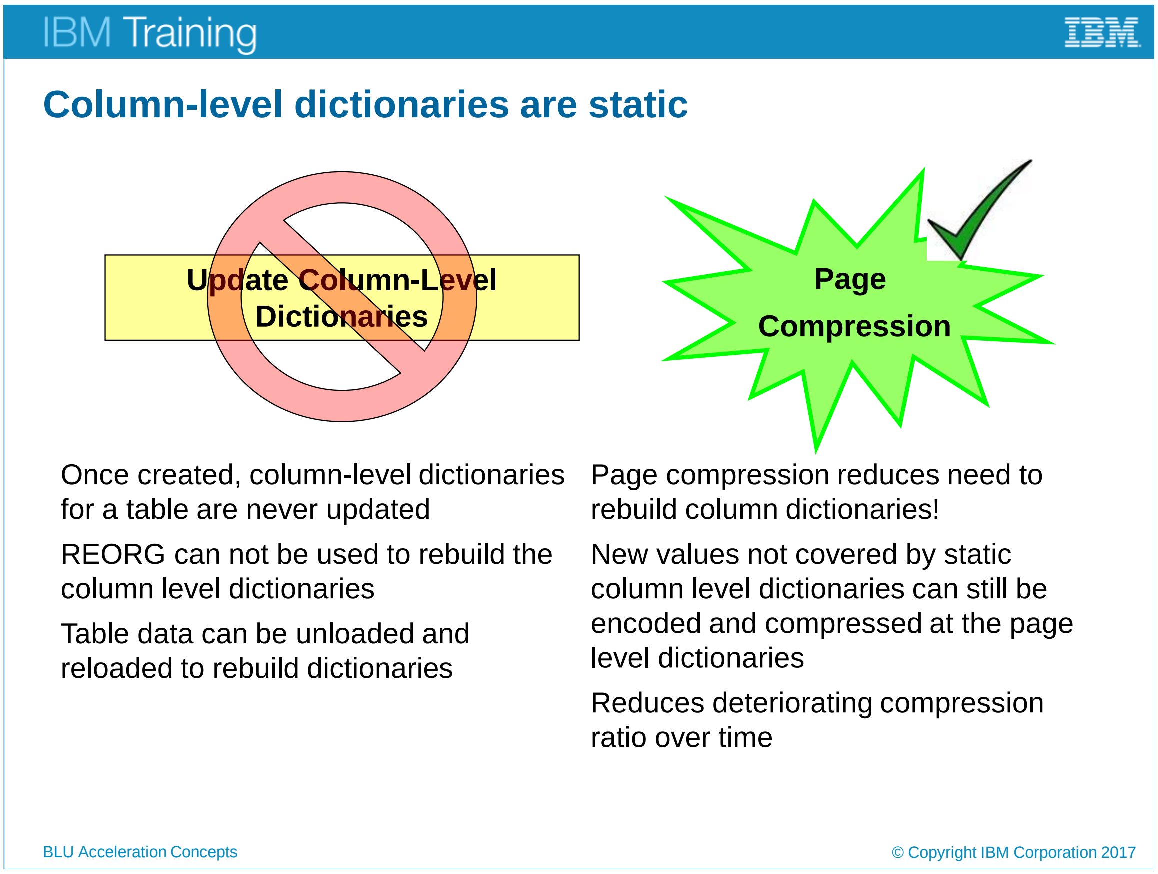
- Exploiting skew in data distribution improves compression ratio
- Very effective since all values in a column have the same data type

BLU Acceleration Concepts © Copyright IBM Corporation 2017

Big Idea #3: BLU uses multiple compression techniques to achieve extreme compression

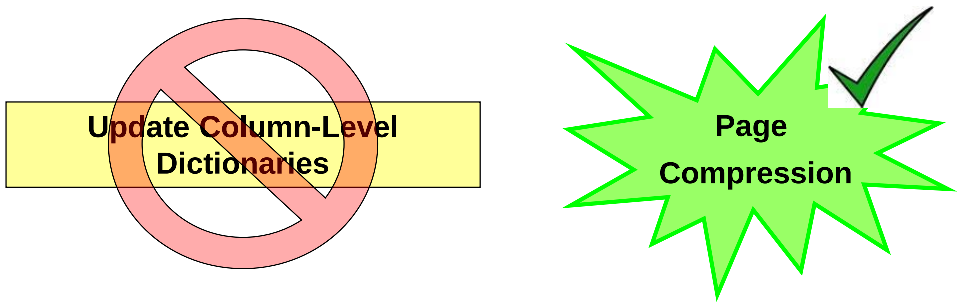
Compression for row-organized tables maps repeated byte patterns across multiple columns to shorter dictionary codes. In contrast, compression for column-organized tables maps entire values to dictionary codes. Since all the column values have the same data type, frequency-based compression can be used to highly compress the data.

The example shows that multiple encoding types can be associated with a single column of a table. DB2 selects the combination of encoding methods for each column based on the column data.



IBM Training

Column-level dictionaries are static



Once created, column-level dictionaries for a table are never updated

REORG can not be used to rebuild the column level dictionaries

Table data can be unloaded and reloaded to rebuild dictionaries

Page compression reduces need to rebuild column dictionaries!

New values not covered by static column level dictionaries can still be encoded and compressed at the page level dictionaries

Reduces deteriorating compression ratio over time

BLU Acceleration Concepts © Copyright IBM Corporation 2017

Column-level dictionaries are static

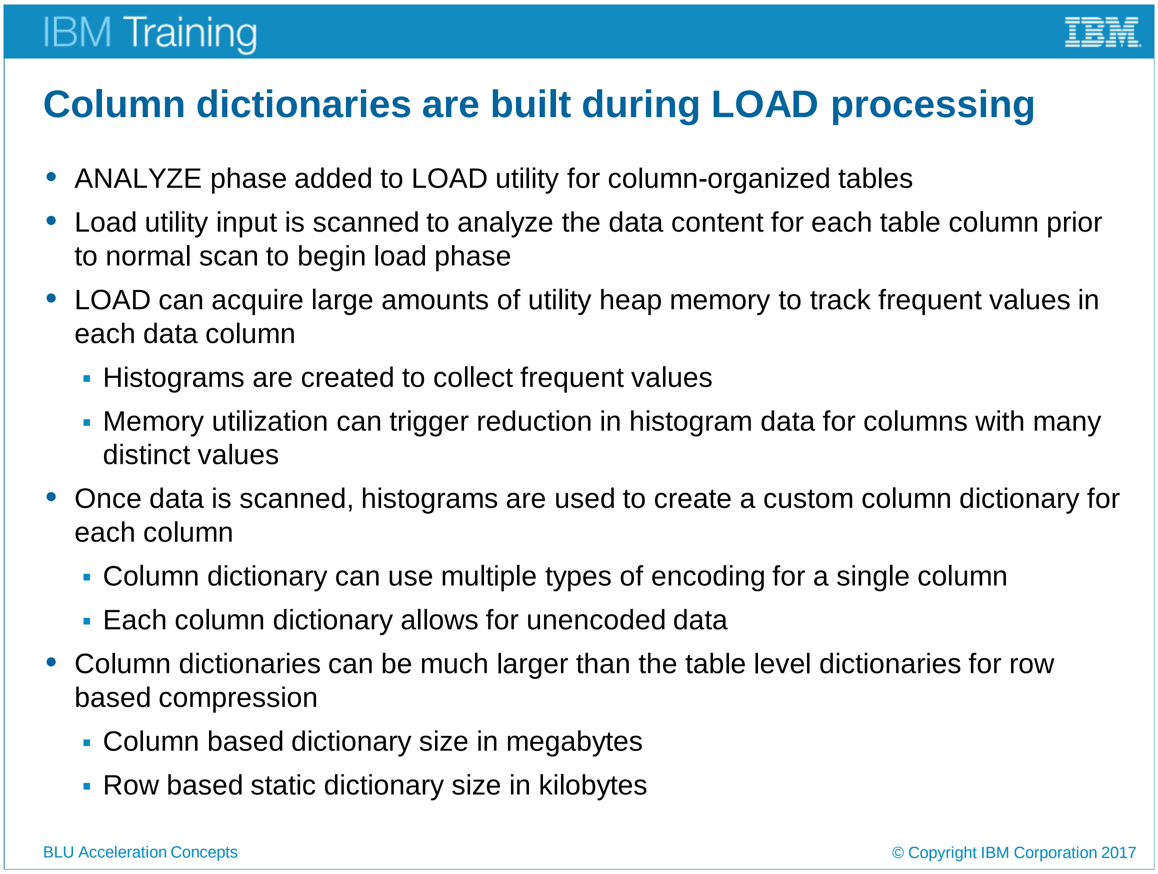
With column-organized table, the column-level dictionaries are created by the LOAD utility as data is loaded. These column-level dictionaries are based on a scan of the LOAD utility input and are never updated once they are built.

The REORG utility does not support standard table reorganization of column-organized tables, so a REORG cannot be used to rebuild the compression dictionaries.

If a column-organized table is extended with new data and the compression ratio drops, the table data could be unloaded and a new LOAD into an empty table could be used to build new column dictionaries.

DB2 does utilize page level dictionaries for column-organized tables. As column data for a set of rows is added to a page, the static column dictionary is used to compress the data. Once the page is nearly full, DB2 will check to see if a page level dictionary, which could handle data values not covered by the column dictionary or some page level adjustment to encoding can be used to improve compression results for the page. In some cases, DB2 could determine that the space needed for the page compression dictionary exceeds the space saved and the page compression would not be used.

The page level dictionary option, is intended to reduce the requirement to rebuild the static column dictionaries. Once a page is filled the page level dictionary will not be changed due to data change activity.



IBM Training

Column dictionaries are built during LOAD processing

- ANALYZE phase added to LOAD utility for column-organized tables
- Load utility input is scanned to analyze the data content for each table column prior to normal scan to begin load phase
- LOAD can acquire large amounts of utility heap memory to track frequent values in each data column
  - Histograms are created to collect frequent values
  - Memory utilization can trigger reduction in histogram data for columns with many distinct values
- Once data is scanned, histograms are used to create a custom column dictionary for each column
  - Column dictionary can use multiple types of encoding for a single column
  - Each column dictionary allows for unencoded data
- Column dictionaries can be much larger than the table level dictionaries for row based compression
  - Column based dictionary size in megabytes
  - Row based static dictionary size in kilobytes

BLU Acceleration Concepts © Copyright IBM Corporation 2017

Column dictionaries are built during LOAD processing

When data is being loaded into a column-organized table, the first phase is the analyze phase, which is unique to column-organized tables.

The analyze phase occurs only if a column compression dictionary needs to be built, which happens during a LOAD REPLACE operation, a LOAD REPLACE RESETDICTIONARY operation, a LOAD REPLACE RESETDICTIONARYONLY operation, or a LOAD INSERT operation (if the column-organized table is empty).

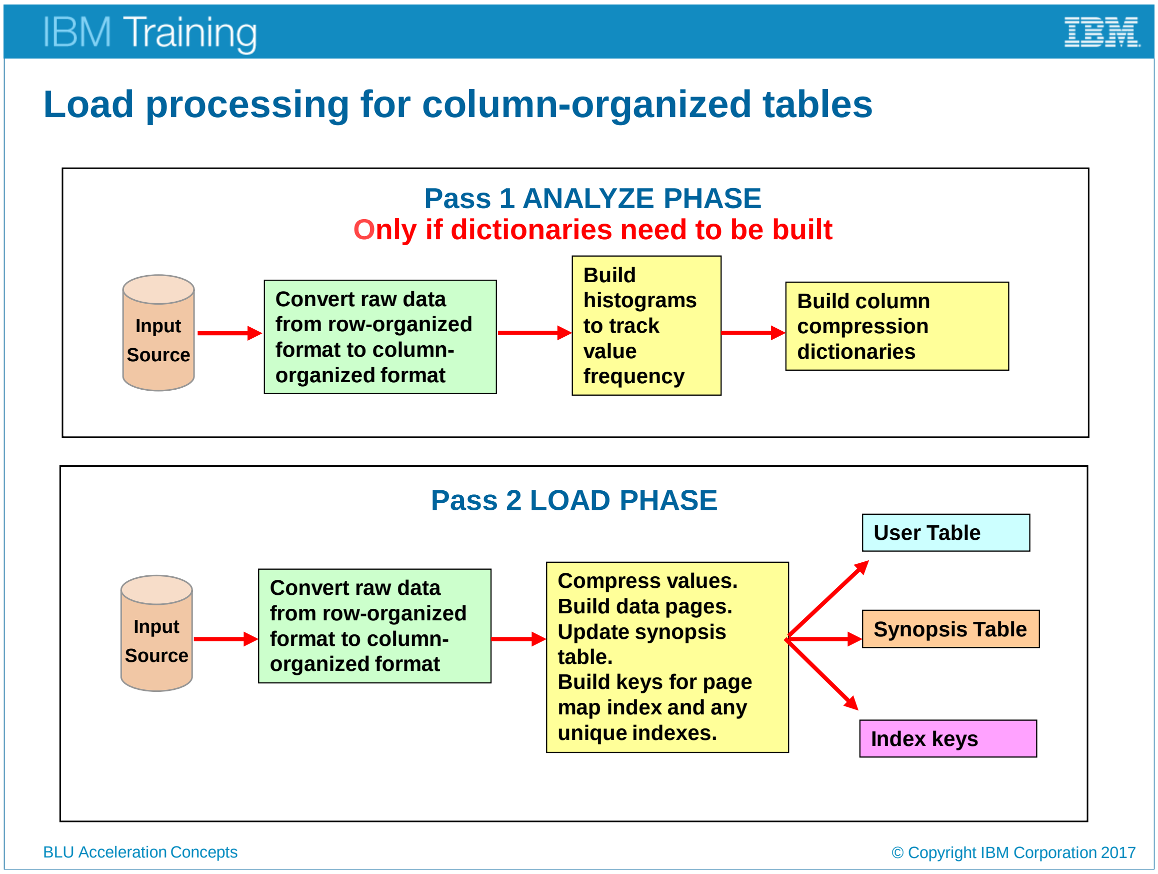
For column-organized tables, this phase is followed by the load, build, and delete phases.

Loading data into column-organized tables is very similar to loading data into row-organized tables, with the following exceptions:

The input data source is processed twice if a column compression dictionary must be built. If the input source can be reopened, it is read twice. If the input source cannot be reopened, its contents are temporarily cached in the load temporary file directory. The default path for load temporary files is located under the instance directory, or in a location that is specified by the TEMPFILES PATH option on the LOAD command.

During the scan for the ANALYZE phase, the LOAD utility can use a large amount of memory in the database utility heap to track the column data values for each column. The goal is to keep column values that are used in many rows so that they can be encoded to save space. DB2 creates histograms for each column to store the values and count occurrences. As utility heap memory becomes full, DB2 will decide how to refine column information in the histograms to reduce memory consumption.

Once the input scan is completed, DB2 reviews the information collected in the histograms for each column to determine the most efficient encoding techniques to use for each column. One factor is the need to limit the size of the column dictionaries.


IBM Training

Load processing for column-organized tables

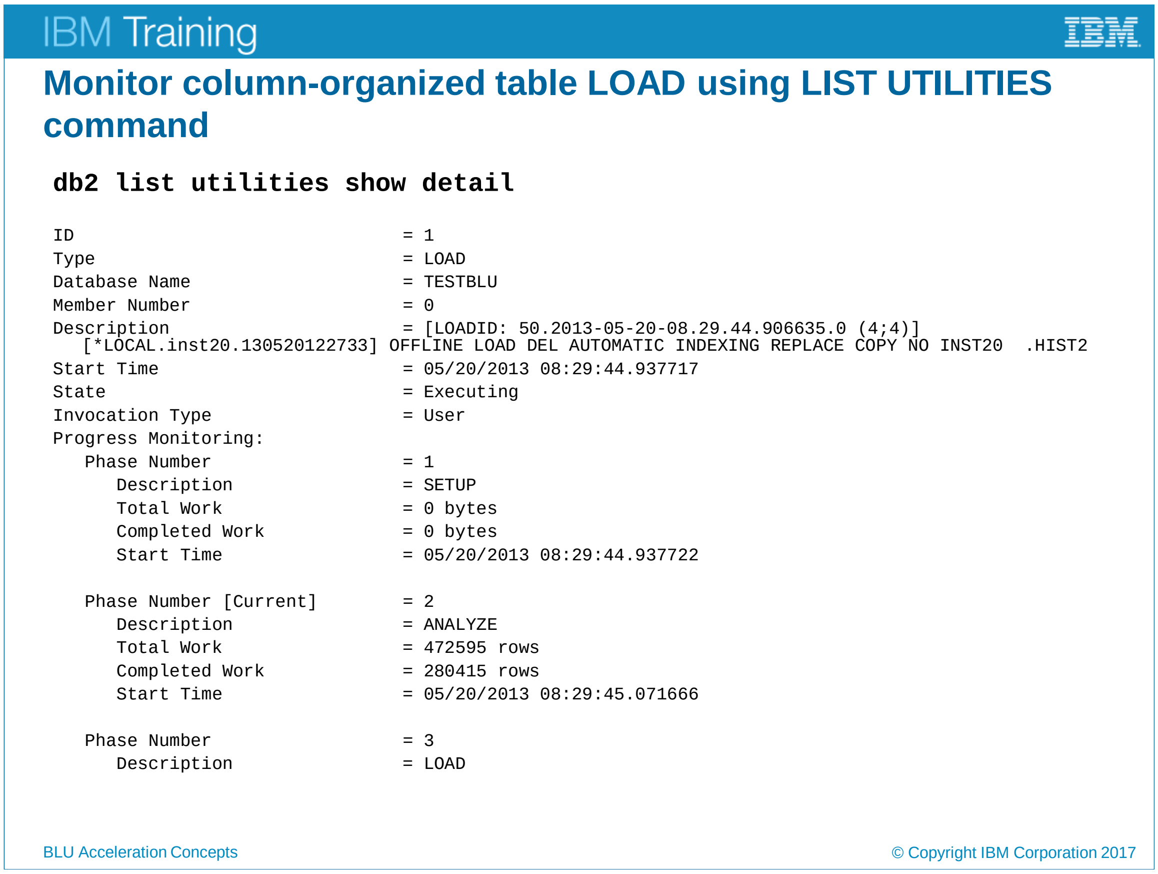
Pass 1 ANALYZE PHASE
Only if dictionaries need to be built
Input Source
Convert raw data from row-organized format to column-organized format
Build histograms to track value frequency
Build column compression dictionaries
Pass 2 LOAD PHASE
Input Source
Convert raw data from row-organized format to column-organized format
Compress values. Build data pages. Update synopsis table. Build keys for page map index and any unique indexes.
User Table
Synopsis Table
Index keys

BLU Acceleration Concepts © Copyright IBM Corporation 2017

Load processing for column-organized tables

Load has three phases that are specific to column-organized tables:

- ANALYZE Phase 1
  - Only applies to column-organized tables if dictionaries needed
  - Load Replace or Load Insert into empty table where KEEPDICTIONARY not specified.
  - Histograms are built to track frequency of data values of all columns
  - Column compression dictionaries are built based on histograms
- LOAD Phase 2 - Modified for column-organized tables
  - Column and page compression dictionaries used to compress data
  - Compressed values written to data pages
  - Synopsis table maintained
  - Keys built for page map index and any unique indexes
- BUILD Phase 3
  - Page map index and any unique indexes are built



IBM Training

Monitor column-organized table LOAD using LIST UTILITIES command

```
db2 list utilities show detail

ID                              = 1
Type                            = LOAD
Database Name                   = TESTBLU
Member Number                   = 0
Description                     = [LOADID: 50.2013-05-20-08.29.44.906635.0 (4;4)]
   [*LOCAL.inst20.130520122733] OFFLINE LOAD DEL AUTOMATIC INDEXING REPLACE COPY NO INST20  .HIST2
Start Time                      = 05/20/2013 08:29:44.937717
State                           = Executing
Invocation Type                 = User
Progress Monitoring:
   Phase Number                 = 1
      Description               = SETUP
      Total Work                = 0 bytes
      Completed Work            = 0 bytes
      Start Time                = 05/20/2013 08:29:44.937722

   Phase Number [Current]       = 2
      Description               = ANALYZE
      Total Work                = 472595 rows
      Completed Work            = 280415 rows
      Start Time                = 05/20/2013 08:29:45.071666

   Phase Number                 = 3
      Description               = LOAD
```

BLU Acceleration Concepts © Copyright IBM Corporation 2017

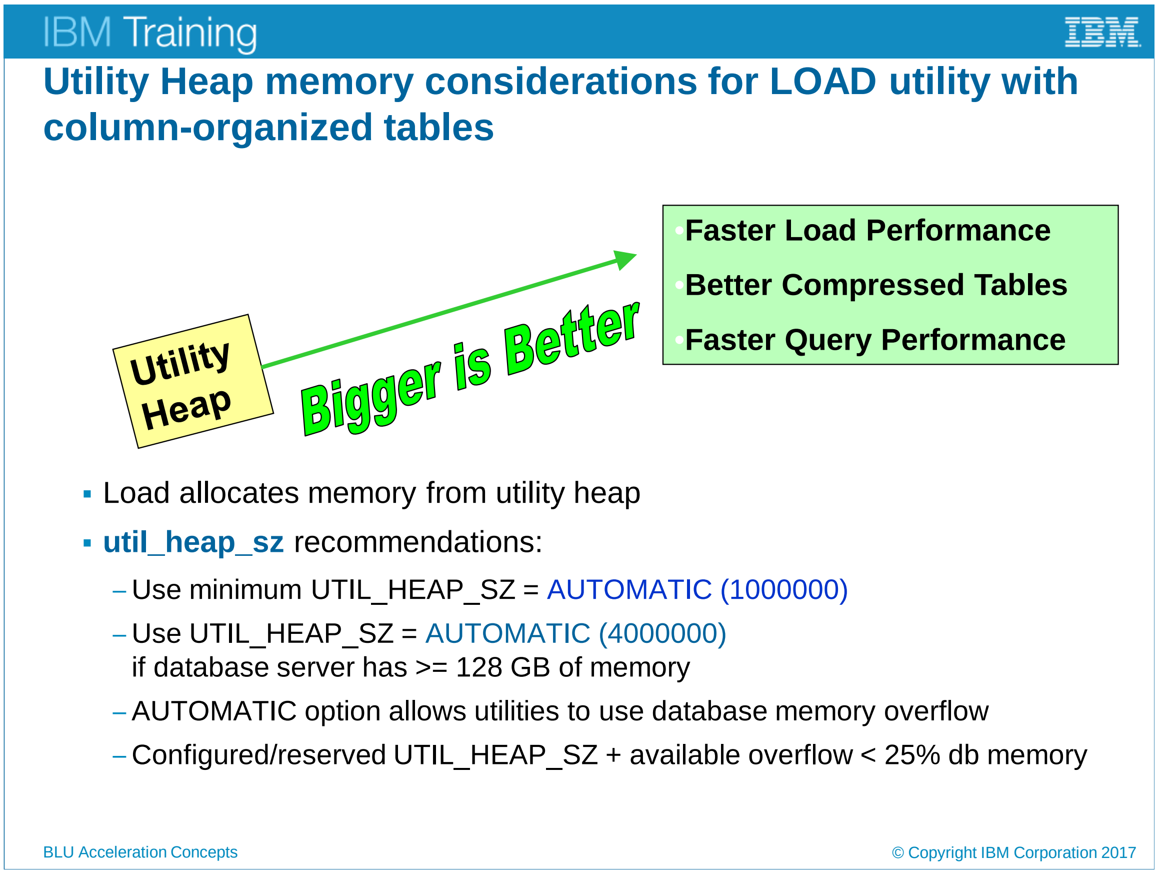
Monitor column-organized table LOAD using LIST UTILITIES command

The slide shows the LIST UTILITIES command output for the LOAD utility with a column-organized table.

In the sample output, the current phase of load processing is the ANALYZE phase, where the column-dictionaries are being built.

Compared to a LOAD for a row-organized table, the LOAD processing for a column-organized table should save considerable time with a reduction in time spent building and updating indexes. The page map index, is page based not row based, so it will be relatively small and easy to build.

If any enforced primary key or unique key constraints are defined on the table, LOAD will need to build those indexes.



IBM Training

Utility Heap memory considerations for LOAD utility with column-organized tables

Utility Heap
Bigger is Better
Faster Load Performance
Better Compressed Tables
Faster Query Performance

- Load allocates memory from utility heap
- **util_heap_sz** recommendations:
  - Use minimum UTIL_HEAP_SZ = AUTOMATIC (1000000)
  - Use UTIL_HEAP_SZ = AUTOMATIC (4000000) if database server has >= 128 GB of memory
  - AUTOMATIC option allows utilities to use database memory overflow
  - Configured/reserved UTIL_HEAP_SZ + available overflow < 25% db memory

BLU Acceleration Concepts © Copyright IBM Corporation 2017

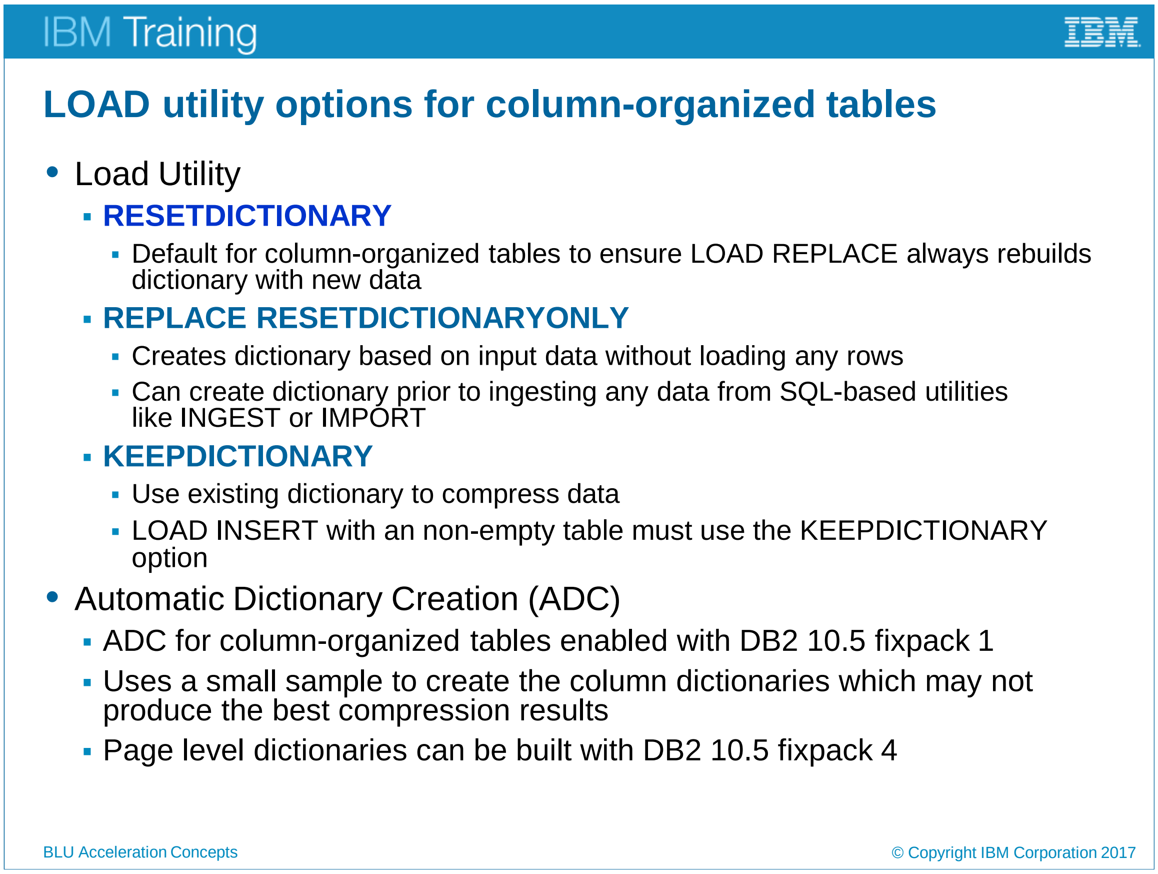
Utility Heap memory considerations for LOAD utility with column-organized tables

Insufficient memory during load utility processing for row-organized tables can reduce load performance but does not affect the efficiency of compression on the tables. After the load completes, there is no long-term negative effect on the tables.

In contrast, insufficient memory during the load ANALYZE phase for column-organized tables could yield less than optimal compressed tables. It is critical to have sufficient utility heap memory, but the memory allocated for utility should be in proportion to the total amount of database server memory so that performance of non-utility operations is not impacted. The performance of the LOAD phase can be improved by additional memory.

Sufficient memory results in faster load performance, better compressed tables, and faster query performance.

- The UTIL_HEAP_SZ (utility heap size) database configuration parameter should be set to at least 1,000,000 pages to address the resource needs of the LOAD command.
- If the database server has at least 128 GB of memory, UTIL_HEAP_SZ should be set to 4,000,000 pages.
- If concurrent load utilities are running, the UTIL_HEAP_SZ value should be increased to accommodate higher memory requirements.
- If memory is scarce, the UTIL_HEAP_SZ value can be increased dynamically only when a load operation is running.



IBM Training

LOAD utility options for column-organized tables

- Load Utility
  - **RESETDICTIONARY**
    - Default for column-organized tables to ensure LOAD REPLACE always rebuilds dictionary with new data
  - **REPLACE RESETDICTIONARYONLY**
    - Creates dictionary based on input data without loading any rows
    - Can create dictionary prior to ingesting any data from SQL-based utilities like INGEST or IMPORT
  - **KEEPDICTIONARY**
    - Use existing dictionary to compress data
    - LOAD INSERT with an non-empty table must use the KEEPDICTIONARY option
- Automatic Dictionary Creation (ADC)
  - ADC for column-organized tables enabled with DB2 10.5 fixpack 1
  - Uses a small sample to create the column dictionaries which may not produce the best compression results
  - Page level dictionaries can be built with DB2 10.5 fixpack 4

BLU Acceleration Concepts © Copyright IBM Corporation 2017

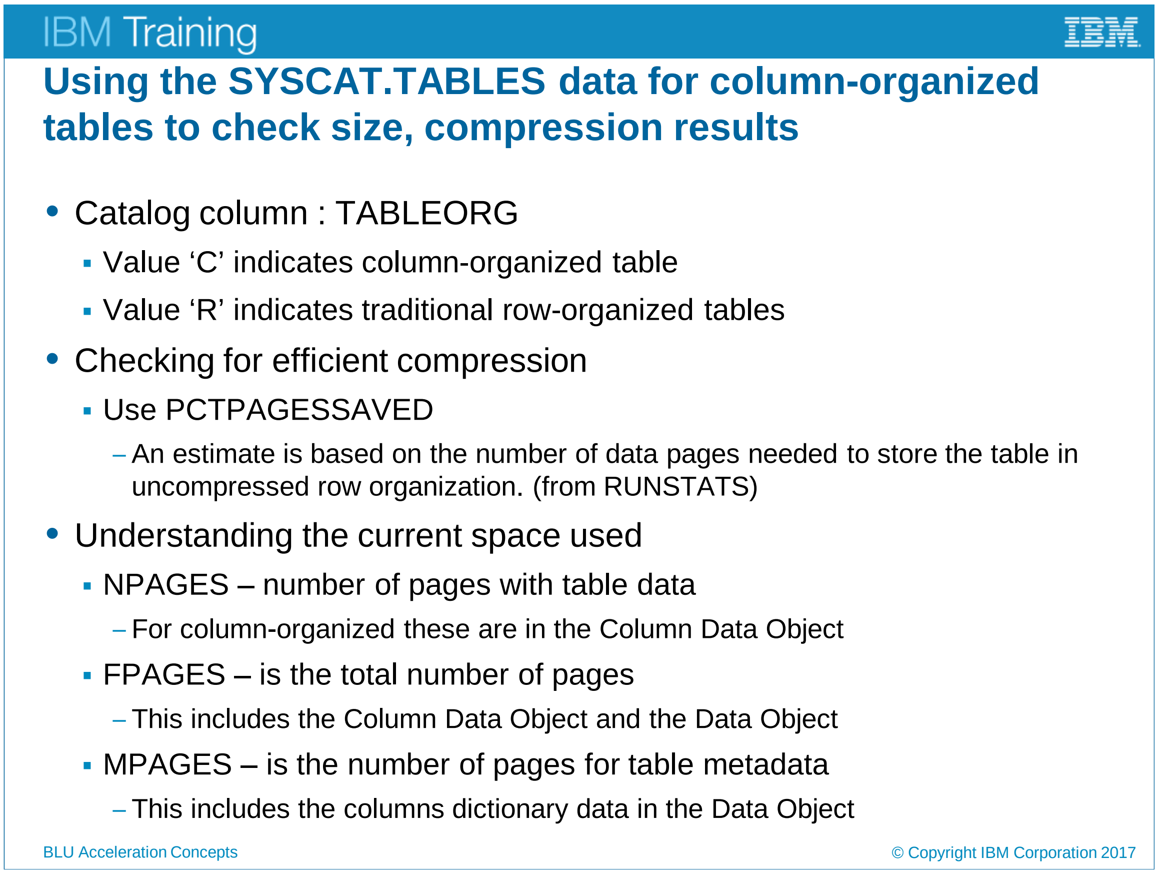
LOAD utility options for column-organized tables

Since the column-dictionaries used for column-organized tables are very dependent on the table data, the default option for loading column-organized tables with a LOAD REPLACE is RESETDICTIONARY, to build new column dictionaries.

The option RESETDICTIONARYONLY creates a column compression dictionary that is based on the input file without loading any rows. You can use this option to create the compression dictionary before you ingest any data by using SQL-based utilities. This option is applicable to column-organized tables only.

The KEEPDICTIONARY option must be used for LOAD INSERT to a table that contains data. It can also be used for LOAD REPLACE to keep the column dictionaries previously created and apply that to the new input without using the ANALYZE phase of processing.

Starting with DB2 10.5 Fix Pack 1, automatic dictionary creation (ADC), can build the column dictionaries for a column-organized table during SQL INSERT processing, which would include IMPORT or INGEST usage. With DB2 10.5 Fix Pack 4 and above, INSERT processing is able to create page-level column dictionary data to improve compression results.


IBM Training

Using the SYSCAT.TABLES data for column-organized tables to check size, compression results

- Catalog column : TABLEORG
  - Value ‘C’ indicates column-organized table
  - Value ‘R’ indicates traditional row-organized tables
- Checking for efficient compression
  - Use PCTPAGESSAVED
    - An estimate is based on the number of data pages needed to store the table in uncompressed row organization. (from RUNSTATS)
- Understanding the current space used
  - NPAGES – number of pages with table data
    - For column-organized these are in the Column Data Object
  - FPAGES – is the total number of pages
    - This includes the Column Data Object and the Data Object
  - MPAGES – is the number of pages for table metadata
    - This includes the columns dictionary data in the Data Object

BLU Acceleration Concepts © Copyright IBM Corporation 2017

Using the SYSCAT.TABLES data for column-organized tables to check size, compression results

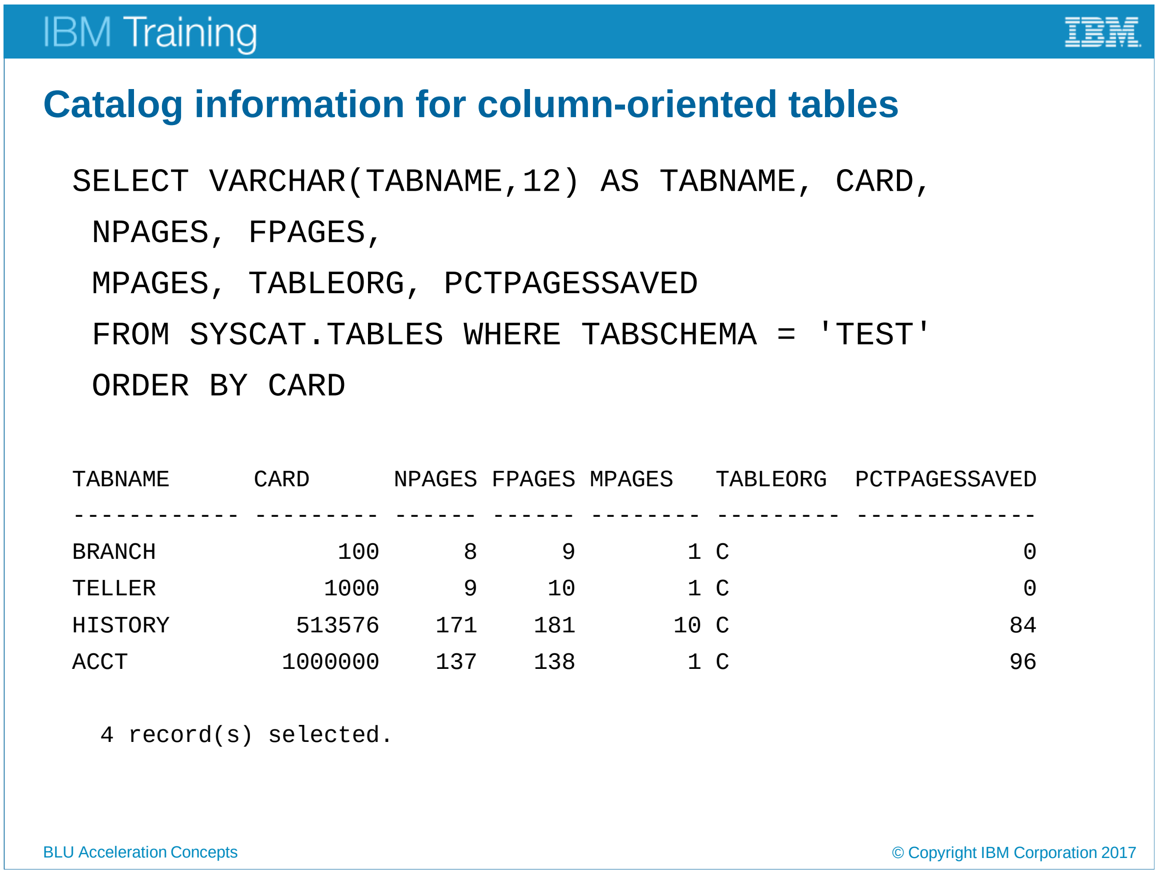
The system catalog view SYSCAT.TABLES can be used to check for efficient compression results with column-organized tables. The value of the column PCTPAGESSAVED provides an estimate of the savings in data page storage for column-organized tables compared to the uncompressed row data. This information is collected by the RUNSTATS utility.

The column TABLEORG in SYSCAT.TABLES will have a value of ‘C’ for column-organized tables.

For column-organized tables, the user table data is stored in a special column organized storage object. The column NPAGES in SYSCAT.TABLES is a count of these pages with table column data.

Since the column dictionaries for column-organized tables can be much larger than the dictionary data stored for row organized tables, the column MPAGES in SYSCAT.TABLES shows the total number of pages used for table metadata, which includes these column dictionaries.

The column FPAGES in SYSCAT.TABLES shows a total page count for column-organized tables which includes both the data and column organized object.


IBM Training

Catalog information for column-oriented tables

```
SELECT VARCHAR(TABNAME,12) AS TABNAME, CARD,
 NPAGES, FPAGES,
 MPAGES, TABLEORG, PCTPAGESSAVED
 FROM SYSCAT.TABLES WHERE TABSCHEMA = 'TEST'
 ORDER BY CARD
```

```
TABNAME      CARD      NPAGES FPAGES MPAGES   TABLEORG  PCTPAGESSAVED
------------ --------- ------ ------ -------- --------- -------------
BRANCH             100      8      9        1 C                     0
TELLER            1000      9     10        1 C                     0
HISTORY         513576    171    181       10 C                    84
ACCT           1000000    137    138        1 C                    96

  4 record(s) selected.
```

BLU Acceleration Concepts © Copyright IBM Corporation 2017

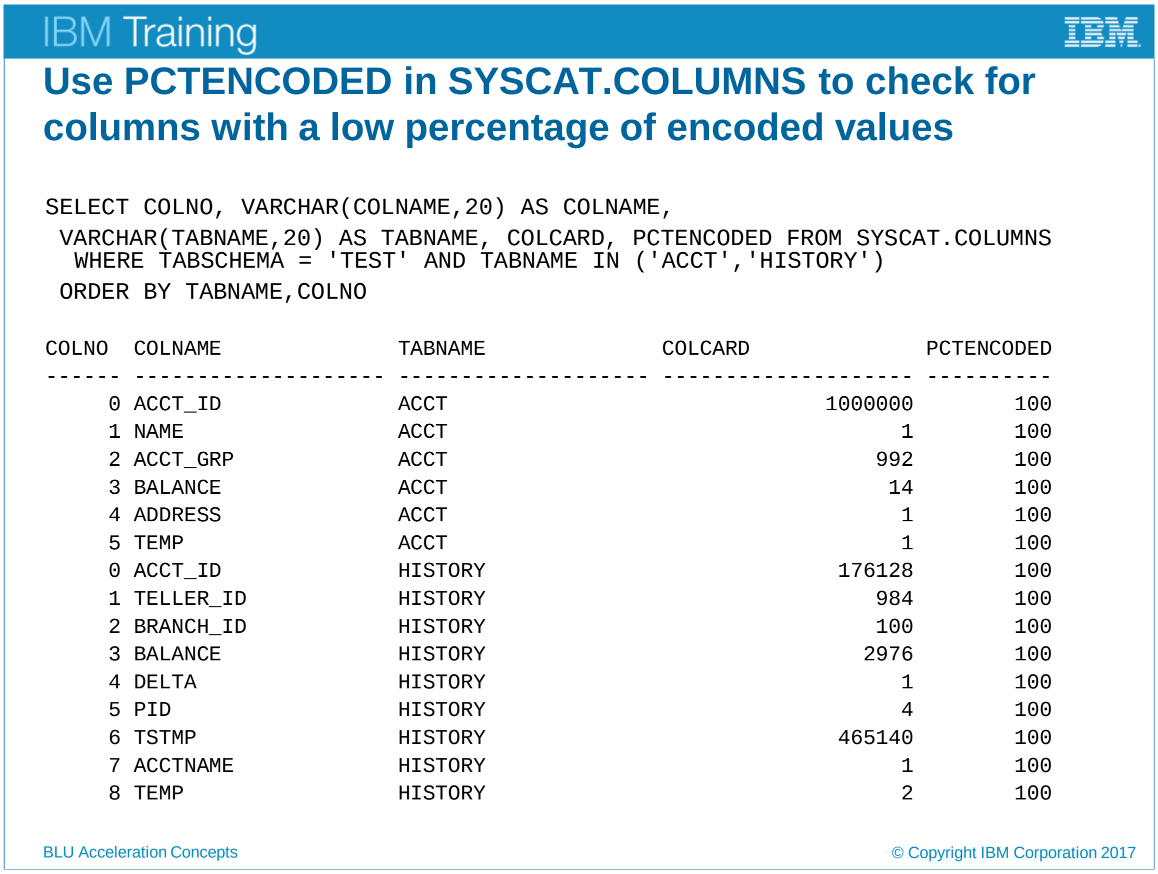
Catalog information for column-oriented tables

The sample SQL query uses the SYSCAT.TABLES view to check the compression results and disk space allocations for several column-organized tables.

Notice that the small tables do not show good compression results. This is caused by the allocation of pages per column in column-organized tables that are not filled with just a few rows of data. Since these tables are small, the lack of space savings for storage is not important.

Some of the catalog statistics that have been used to evaluate compression results for row-organized tables do not apply to column-organized tables. Some examples are AVGCOMPRESSEDROWSIZE and PCTROWSCOMPRESSED.

We will discuss later why it is still recommended to create the small tables that will be joined with larger tables as column-organized to get the benefits of efficient join processing.



IBM Training

Use PCTENCODED in SYSCAT.COLUMNS to check for columns with a low percentage of encoded values

```
SELECT COLNO, VARCHAR(COLNAME,20) AS COLNAME,
 VARCHAR(TABNAME,20) AS TABNAME, COLCARD, PCTENCODED FROM SYSCAT.COLUMNS
  WHERE TABSCHEMA = 'TEST' AND TABNAME IN ('ACCT','HISTORY')
 ORDER BY TABNAME,COLNO

COLNO  COLNAME              TABNAME              COLCARD              PCTENCODED
------ -------------------- -------------------- -------------------- ----------
     0 ACCT_ID              ACCT                              1000000        100
     1 NAME                 ACCT                                    1        100
     2 ACCT_GRP             ACCT                                  992        100
     3 BALANCE              ACCT                                   14        100
     4 ADDRESS              ACCT                                    1        100
     5 TEMP                 ACCT                                    1        100
     0 ACCT_ID              HISTORY                            176128        100
     1 TELLER_ID            HISTORY                               984        100
     2 BRANCH_ID            HISTORY                               100        100
     3 BALANCE              HISTORY                              2976        100
     4 DELTA                HISTORY                                 1        100
     5 PID                  HISTORY                                 4        100
     6 TSTMP                HISTORY                            465140        100
     7 ACCTNAME             HISTORY                                 1        100
     8 TEMP                 HISTORY                                 2        100
```

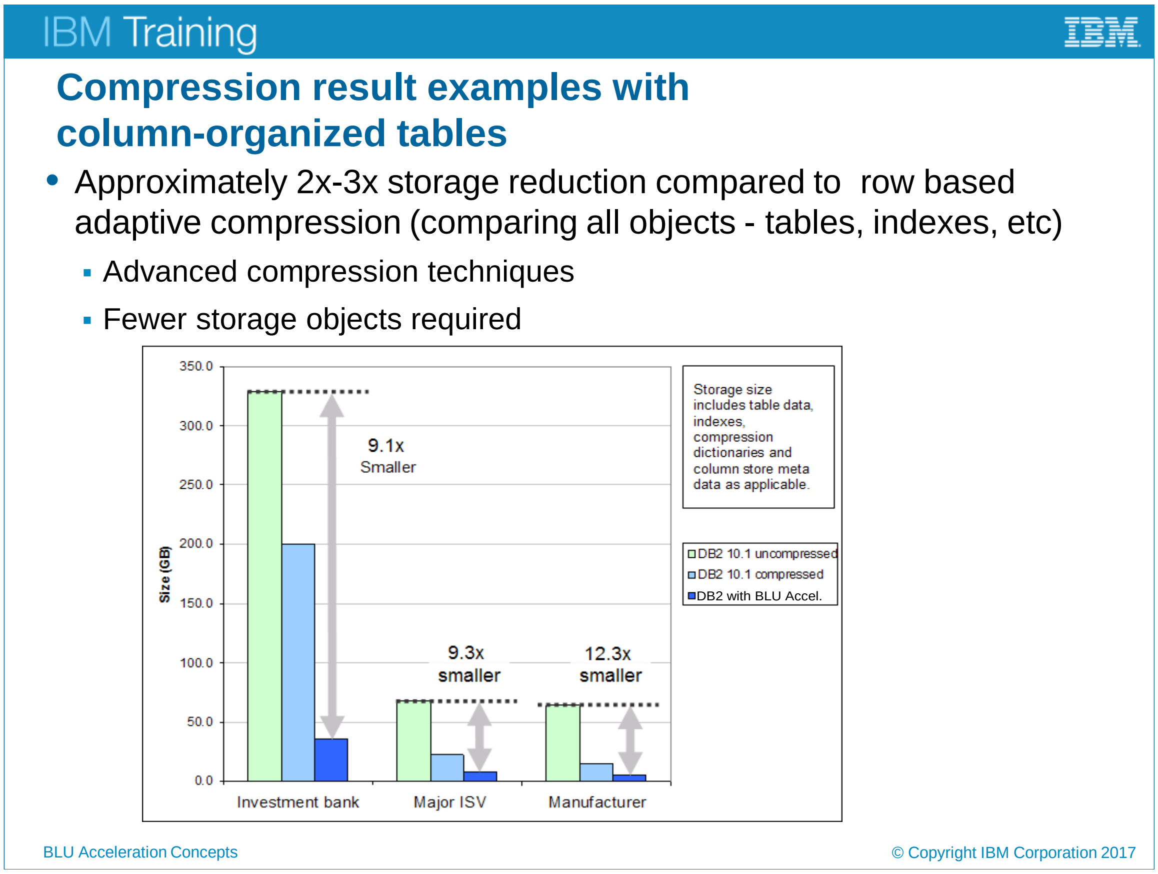
BLU Acceleration Concepts

© Copyright IBM Corporation 2017

Use PCTENCODED in SYSCAT.COLUMNS to check for columns with a low percentage of encoded values

The sample SQL query shows how the SYSCAT.COLUMNS view can be used to check if each column dictionary of a column-organized table is efficiently compressing the data values for that column of the table.

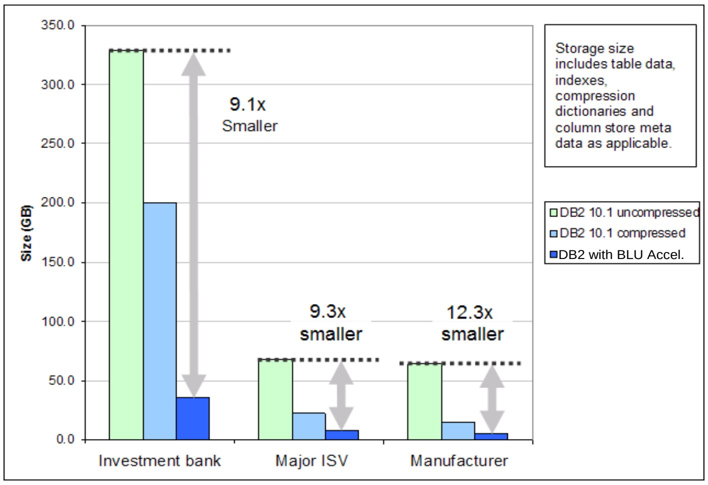
The column PCTENCODED in SYSCAT.COLUMNS shows the percentage of values that are encoded as a result of compression for a column in a column-organized table. If the encoding techniques stored in the column dictionary of a column do not match a data value in the column, the data can be stored as unencoded. If a large percentage of the values in a column are unencoded, it may be the data used to build the column dictionary does not match the current data for the column. Page level dictionaries can handle some new data values by creating a page level dictionary that provides encoding not present in the static column dictionary.



IBM Training

Compression result examples with column-organized tables

- Approximately 2x-3x storage reduction compared to row based adaptive compression (comparing all objects - tables, indexes, etc)
  - Advanced compression techniques
  - Fewer storage objects required



BLU Acceleration Concepts

© Copyright IBM Corporation 2017

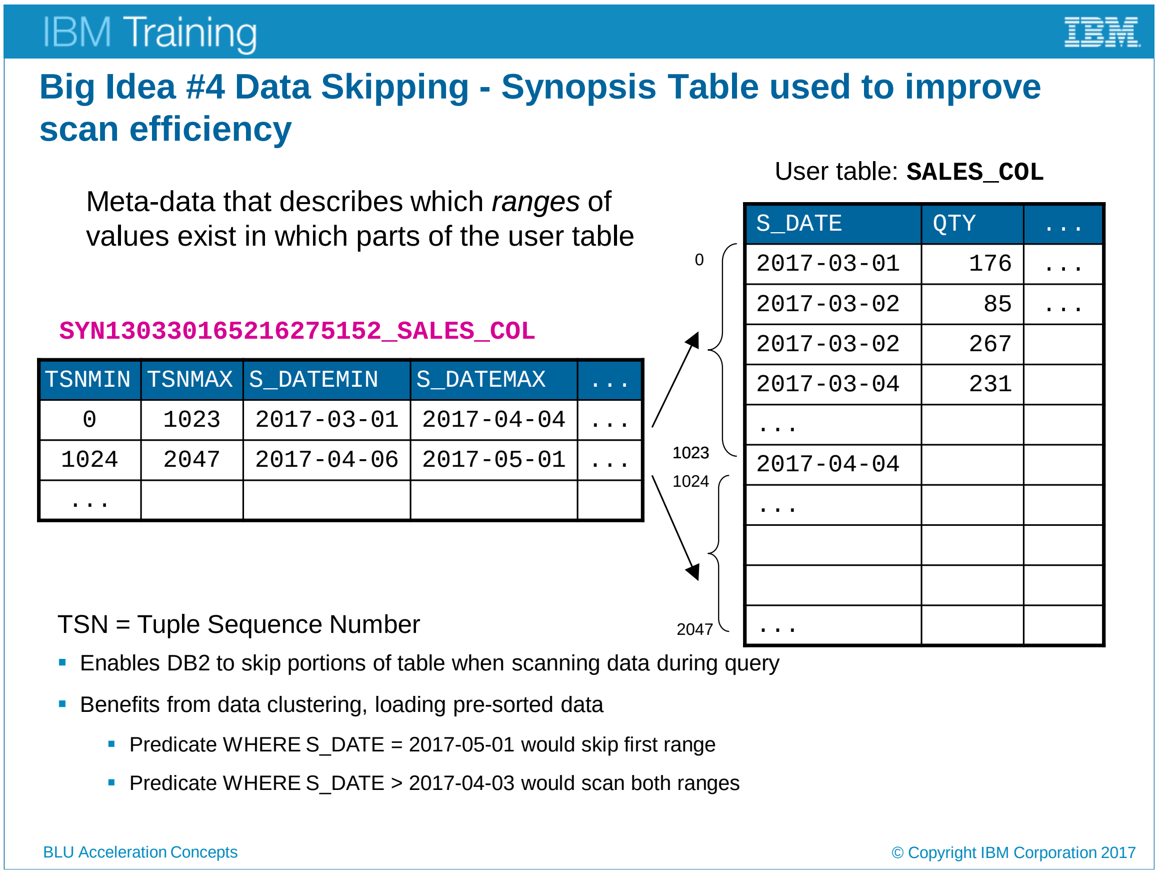
Compression result examples with column-organized tables

The slide shows the reductions is storage requirements for several different types of customer data using column-organized tables.

The three sets of bars show:

- The space requirements for uncompressed tables using DB2 10.1
- The space required for compressed table using DB2 10.1 adaptive compression.
- The space required for compressed column-organized tables using DB2 10.5

A portion of the space saved using column-organized tables is the space that was used for additional objects like indexes for the row-organized tables.



IBM Training

Big Idea #4 Data Skipping - Synopsis Table used to improve scan efficiency

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	2017-03-01	2017-04-04	...
1024	2047	2017-04-06	2017-05-01	...
...				

User table: **SALES_COL**

S_DATE	QTY	...
2017-03-01	176	...
2017-03-02	85	...
2017-03-02	267	
2017-03-04	231	
...		
2017-04-04		
...		
...		

TSN = Tuple Sequence Number

- Enables DB2 to skip portions of table when scanning data during query
- Benefits from data clustering, loading pre-sorted data
  - Predicate WHERE S_DATE = 2017-05-01 would skip first range
  - Predicate WHERE S_DATE > 2017-04-03 would scan both ranges

BLU Acceleration Concepts © Copyright IBM Corporation 2017

Big Idea #4 Data Skipping - Synopsis Table used to improve scan efficiency

It is common to create indexes on row-organized tables to provide more efficient access to various subsets of a table and to avoid table scans which consume system I/O resources and memory. To support different types of table access, a series of indexes may be created which require storage space and increase the overhead of loading and changing table data.

Column-organized tables can have a primary key or unique key defined but do not allow user created indexes to be created using the CREATE INDEX statement. DB2 does automatically maintain a synopsis table for column-organized tables. Unlike standard indexes, the synopsis table describes a range of rows rather than a single row.

The slide shows an example of a table SALES_COL, which contains a date column named S_DATE. As data rows are added to the table SALES_COL, DB2 records the minimum and maximum values for the column S_DATE for a range of rows.

The first entry in the synopsis table covers the first 1024 rows or TSNs. For the column S_DATE the synopsis table has two columns that indicate a range of data values from ‘2017-03-01’ to ‘’2017-04-04’.

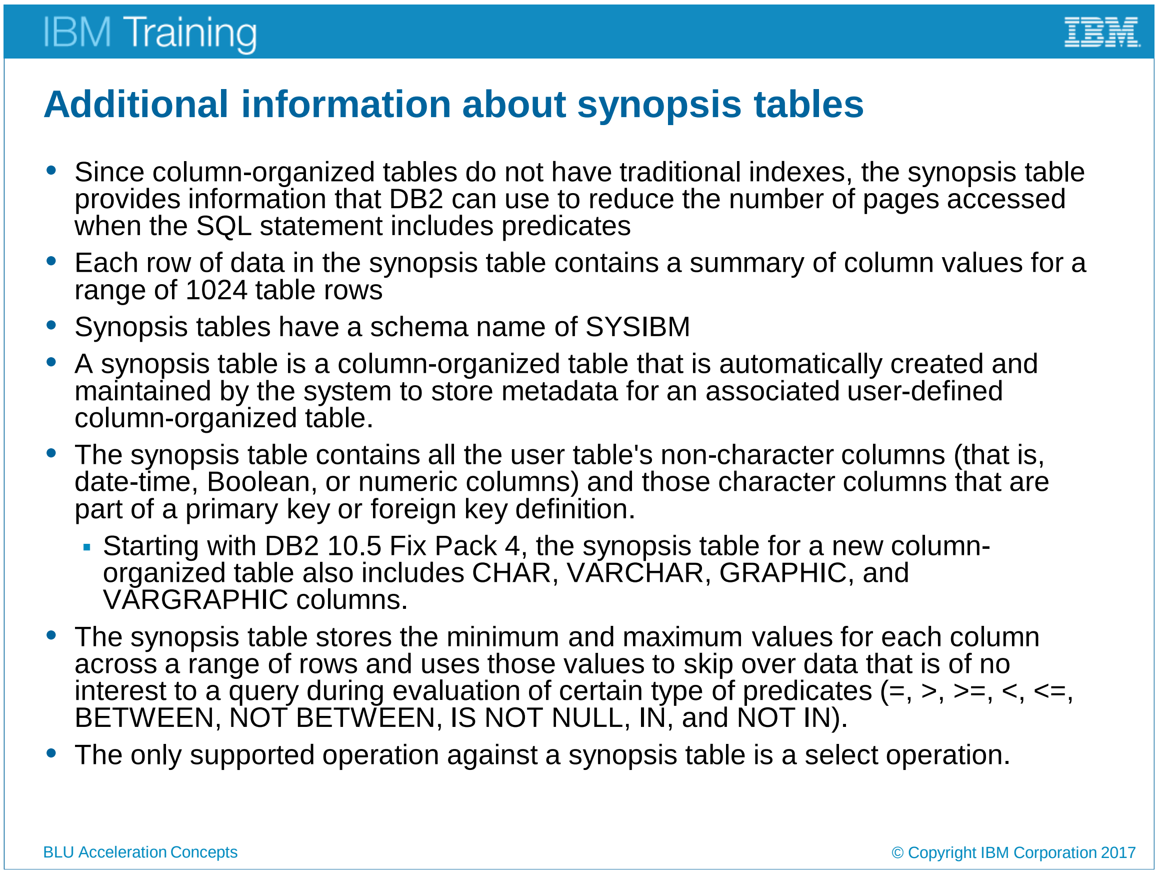
If an application requests data from the table SALES-COL and includes the predicate WHERE S_DATE = ‘2017-05-01’, DB2 could use the information in the synopsis table to skip reading the first 1025 TSNs, since the predicate is outside the range contained in those rows. The second range, TSNs 1024 to 2047 would be scanned.

If an application selects data from the table using the predicate ‘S-DATE > ‘2017-04-03’, the data for the S_DATE column in the synopsis table would indicate that both of the TSN ranges shown would need to be scanned.

The information in the synopsis table is most valuable when the table data is clustered using the column referenced by the predicate. In the previous example, if the data is loaded in S_DATE column sequence, then queries that include predicates for certain ranges of S_DATE values can use the synopsis table to skip large portions of the table data.

If a table has rows in a random sequence for a particular column, the synopsis table will be less useful in allowing ranges of rows to be skipped during scans.

This is similar to having an unclustered index on a table. In order to retrieve a large set of result rows, many pages need to be accessed and DB2 may decide to not use the index.



IBM Training

Additional information about synopsis tables

- Since column-organized tables do not have traditional indexes, the synopsis table provides information that DB2 can use to reduce the number of pages accessed when the SQL statement includes predicates
- Each row of data in the synopsis table contains a summary of column values for a range of 1024 table rows
- Synopsis tables have a schema name of SYSIBM
- A synopsis table is a column-organized table that is automatically created and maintained by the system to store metadata for an associated user-defined column-organized table.
- The synopsis table contains all the user table's non-character columns (that is, date-time, Boolean, or numeric columns) and those character columns that are part of a primary key or foreign key definition.
  - Starting with DB2 10.5 Fix Pack 4, the synopsis table for a new column-organized table also includes CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC columns.
- The synopsis table stores the minimum and maximum values for each column across a range of rows and uses those values to skip over data that is of no interest to a query during evaluation of certain type of predicates (=, >, >=, <, <=, BETWEEN, NOT BETWEEN, IS NOT NULL, IN, and NOT IN).
- The only supported operation against a synopsis table is a select operation.

BLU Acceleration Concepts © Copyright IBM Corporation 2017

Additional information about synopsis tables

The synopsis table includes information about multiple columns for a range of 1024 data rows, so it will be much smaller than the table that it describes.

Synopsis tables are automatically created and maintained by DB2. You do not select which columns are summarized within the synopsis table. As data rows are loaded or inserted, DB2 will create the matching entries in the synopsis table.

The synopsis table is stored internally as a column-organized table. DB2 is able to access the information about selected columns very efficiently.

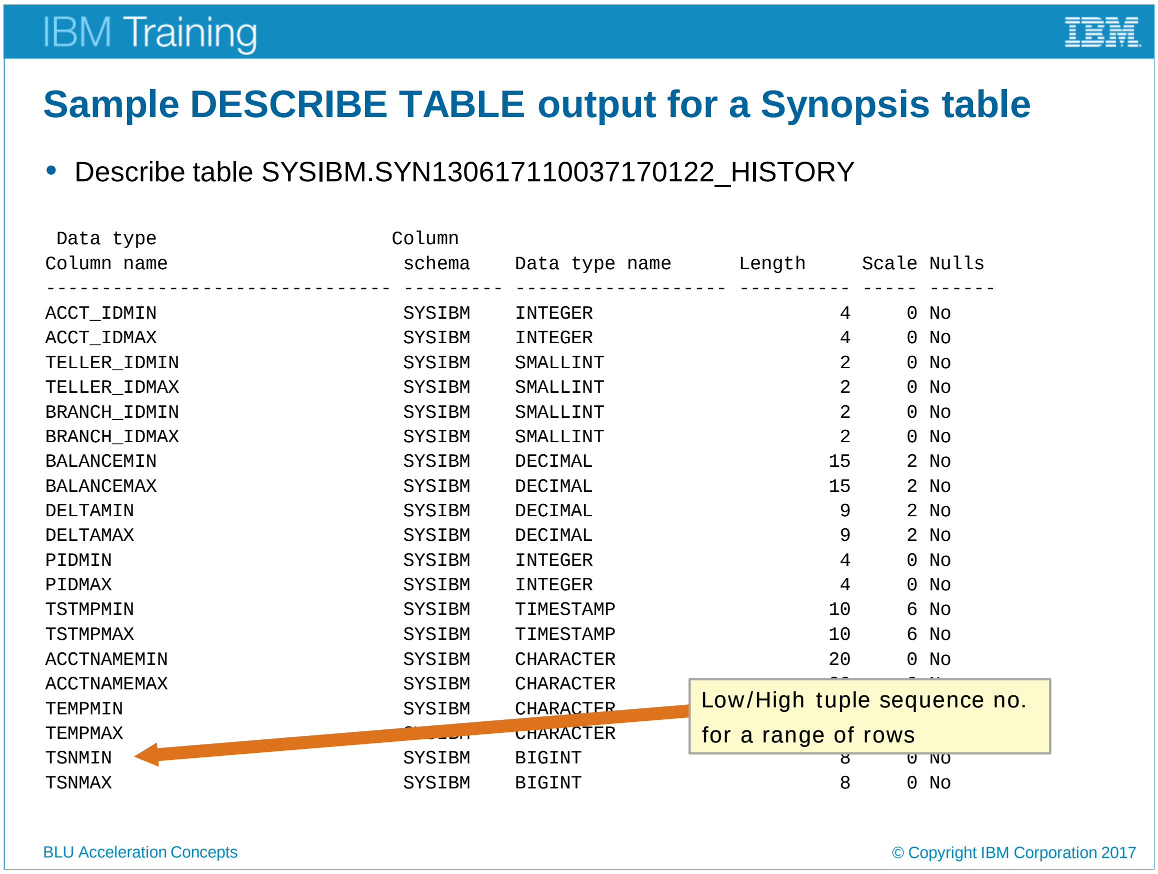
DB2 created a generated name for synopsis tables than begin with a prefix of ‘SYN’ and use the table name as suffix. DB2 uses the schema name of SYSIBM rather than using the schema of the base table. A synopsis table is a column-organized table that is automatically created and maintained by the system to store metadata for an associated user-defined column-organized table.

The synopsis table contains all the user table's non-character columns (that is, datetime, Boolean, or numeric columns) and those character columns that are part of a primary key or foreign key definition.

Starting with Version 10.5 Fix Pack 4, the synopsis table for a new column-organized table also includes CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC columns.

The synopsis table stores the minimum and maximum values for each column across a range of rows and uses those values to skip over data that is of no interest to a query during evaluation of certain predicate types (=, >, >=, <, <=, BETWEEN, NOT BETWEEN, IS NOT NULL, IN, NOT IN, and LIKE).

The only supported operation against a synopsis table is a select operation.



IBM Training

Sample DESCRIBE TABLE output for a Synopsis table

- Describe table SYSIBM.SYN130617110037170122_HISTORY

```
 Data type                     Column
Column name                     schema    Data type name      Length     Scale Nulls
------------------------------- --------- ------------------- ---------- ----- ------
ACCT_IDMIN                      SYSIBM    INTEGER                      4     0 No
ACCT_IDMAX                      SYSIBM    INTEGER                      4     0 No
TELLER_IDMIN                    SYSIBM    SMALLINT                     2     0 No
TELLER_IDMAX                    SYSIBM    SMALLINT                     2     0 No
BRANCH_IDMIN                    SYSIBM    SMALLINT                     2     0 No
BRANCH_IDMAX                    SYSIBM    SMALLINT                     2     0 No
BALANCEMIN                      SYSIBM    DECIMAL                     15     2 No
BALANCEMAX                      SYSIBM    DECIMAL                     15     2 No
DELTAMIN                        SYSIBM    DECIMAL                      9     2 No
DELTAMAX                        SYSIBM    DECIMAL                      9     2 No
PIDMIN                          SYSIBM    INTEGER                      4     0 No
PIDMAX                          SYSIBM    INTEGER                      4     0 No
TSTMPMIN                        SYSIBM    TIMESTAMP                   10     6 No
TSTMPMAX                        SYSIBM    TIMESTAMP                   10     6 No
ACCTNAMEMIN                     SYSIBM    CHARACTER                   20     0 No
ACCTNAMEMAX                     SYSIBM    CHARACTER
TEMPMIN                         SYSIBM    CHARACTER
TEMPMAX                                   CHARACTER
TSNMIN                          SYSIBM    BIGINT                       8     0 No
TSNMAX                          SYSIBM    BIGINT                       8     0 No
```

Low/High tuple sequence no. for a range of rows

BLU Acceleration Concepts © Copyright IBM Corporation 2017

Sample DESCRIBE TABLE output for a Synopsis table

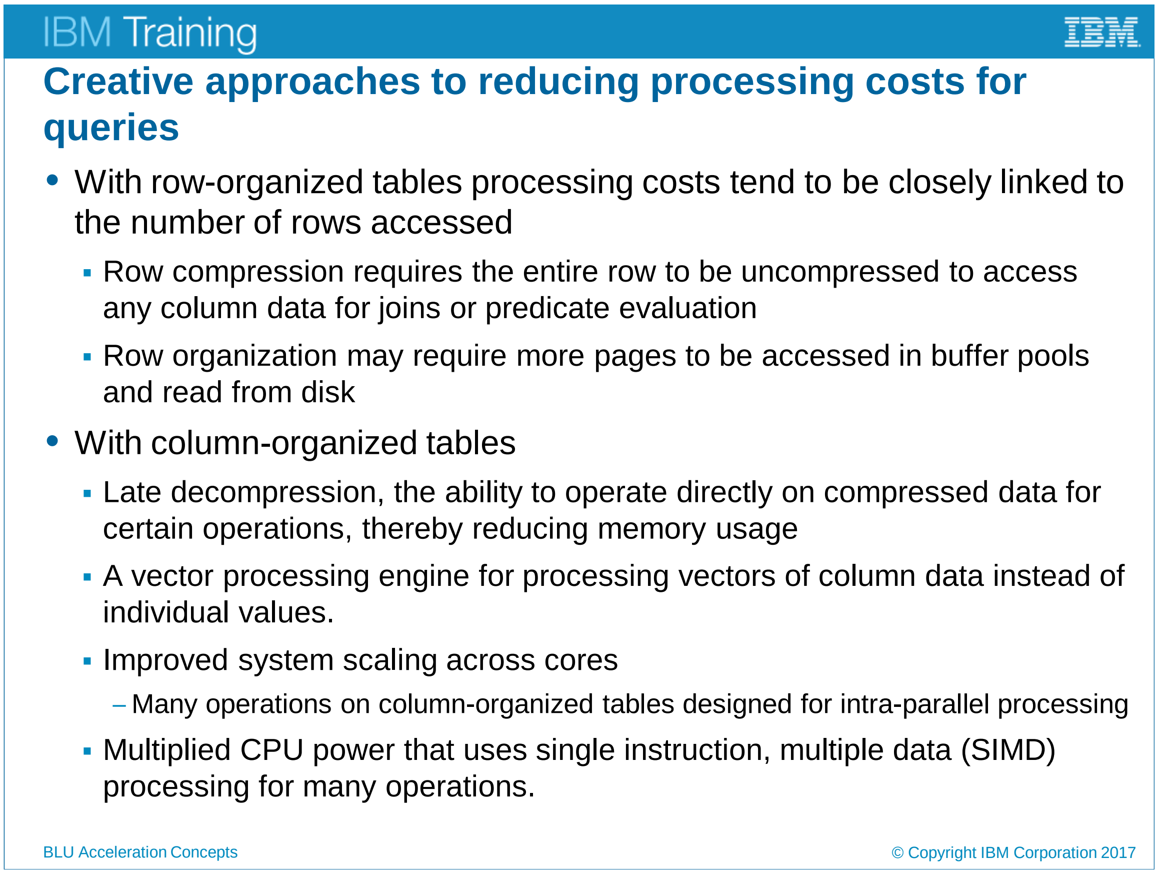
Assume a column-organized table is created with the following CREATE TABLE statement:

```
CREATE TABLE HISTORY
      (ACCT_ID          INTEGER         NOT NULL,
TELLER_ID        SMALLINT        NOT NULL,
BRANCH_ID        SMALLINT        NOT NULL,
BALANCE          DECIMAL(15,2)   NOT NULL,
DELTA            DECIMAL(9,2)    NOT NULL,
PID              INTEGER         NOT NULL,
TSTMP            TIMESTAMP       NOT NULL WITH DEFAULT,
ACCTNAME         CHAR(20)        NOT NULL,
TEMP             CHAR(6)         NOT NULL)
ORGANIZE BY COLUMN
IN TSROWD INDEX IN TSROWI;
```

The DESCRIBE TABLE sample output shows the column names used for a synopsis table that would be automatically created for that table.

Notice that the column names are prefixed with the names of the base table column and suffixed by ‘MIN’ or ‘MAX’ and have the same data type as the base table column.

There is also a pair of columns named ‘TSNMIN’ and ‘TSNMAX’ to store the starting and ending TSN ids for the set of data rows.



IBM Training

Creative approaches to reducing processing costs for queries

- With row-organized tables processing costs tend to be closely linked to the number of rows accessed
  - Row compression requires the entire row to be uncompressed to access any column data for joins or predicate evaluation
  - Row organization may require more pages to be accessed in buffer pools and read from disk
- With column-organized tables
  - Late decompression, the ability to operate directly on compressed data for certain operations, thereby reducing memory usage
  - A vector processing engine for processing vectors of column data instead of individual values.
  - Improved system scaling across cores
    - Many operations on column-organized tables designed for intra-parallel processing
  - Multiplied CPU power that uses single instruction, multiple data (SIMD) processing for many operations.

BLU Acceleration Concepts © Copyright IBM Corporation 2017

Creative approaches to reducing processing costs for queries

With row-organized tables we tend to associate costs with each row accessed by a query. With row based compression, the entire data row needs to be uncompressed before any predicate can be applied to a column value or perform a join with another table. In some cases, an index could be used to bypass the need to access the full data row.

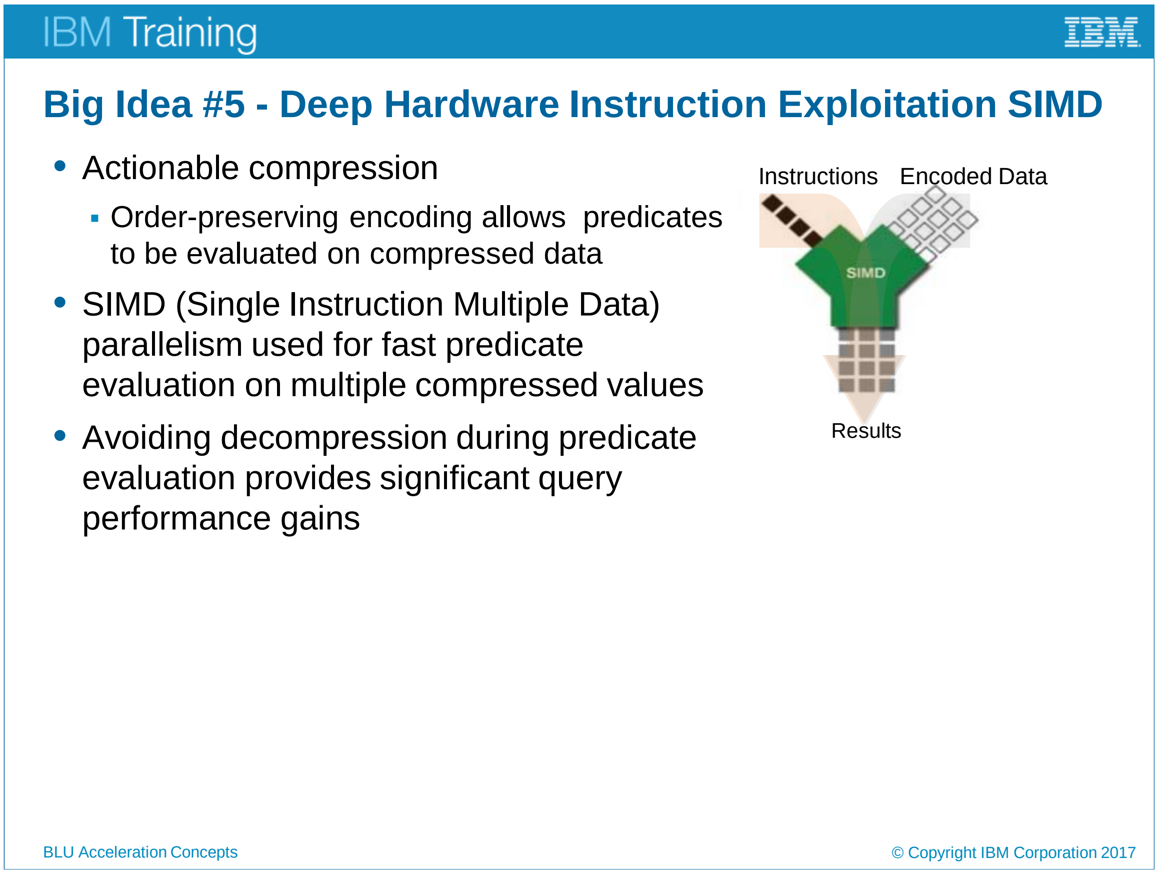
To access one million rows of data, the data pages containing the full rows will need to be accessed in a buffer pool and possibly read from disk.

The design of column-organized tables uses a technique referred to as late decompression, where some predicates and operations like joins can operate directly on the compressed form of the column value.

Column-organized processing also utilizes a technique of vector processing for column data rather than performing operations once per data row.

The new routines that were developed to support access to column-organized tables were designed to drive parallel processing by multiple CPU cores.

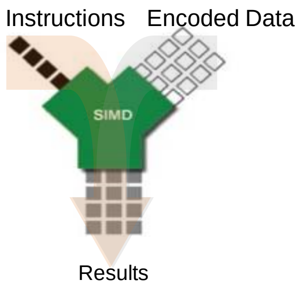
In some cases DB2 leveraged hardware machine instructions termed SIMD for Single Instruction with Multiple Data that improve the efficiency of handling sets of data instead of operating on each single column value.



IBM Training

Big Idea #5 - Deep Hardware Instruction Exploitation SIMD

- Actionable compression
  - Order-preserving encoding allows predicates to be evaluated on compressed data
- SIMD (Single Instruction Multiple Data) parallelism used for fast predicate evaluation on multiple compressed values
- Avoiding decompression during predicate evaluation provides significant query performance gains



BLU Acceleration Concepts

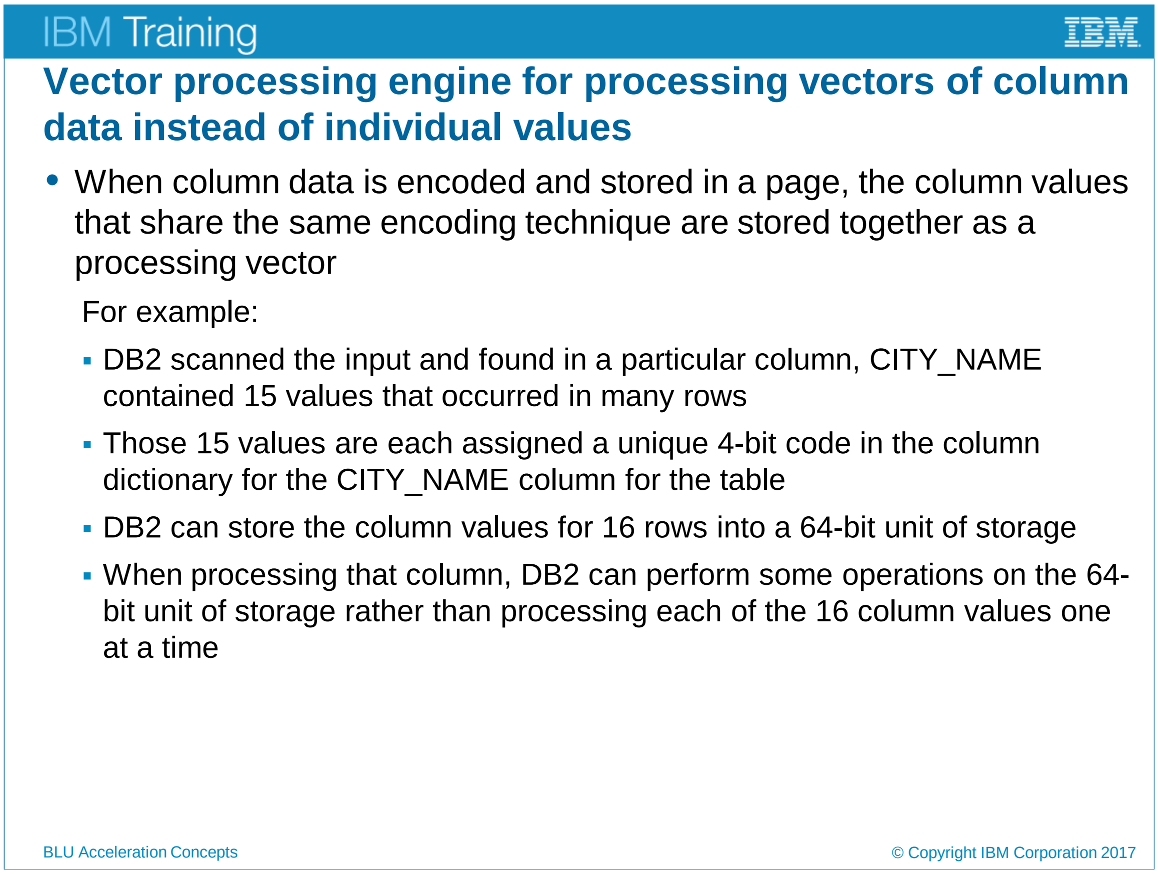
© Copyright IBM Corporation 2017

Big Idea #5 - Deep Hardware Instruction Exploitation SIMD

The fifth big idea used for DB2 BLU Acceleration is the use of special hardware instructions to work on multiple data elements with a single instruction. This is known as Single Instruction Multiple Data or SIMD. There are special instructions available on various hardware platform to accomplish this. DB2 includes special BLU Acceleration code to do this. Currently DB2 can put as much data as 128 bits into a SIMD register.

The implementation for column based compression allows ‘actionable compression’, meaning a number of operations can use the compressed form of a column value directly. The compression technique is ‘order preserving’ meaning DB2 can perform operations like checking ‘GREATER THAN’ and ‘LESS THAN’, not just ‘EQUAL TO’ comparisons on the compressed column values.

DB2 can use a single machine instruction to multiply the power of the CPU, by getting results of all those data values packed into the register. This means we can use the power of the CPUs while performing scans as well as joins and aggregations.



IBM Training

Vector processing engine for processing vectors of column data instead of individual values

- When column data is encoded and stored in a page, the column values that share the same encoding technique are stored together as a processing vector

  For example:

  - DB2 scanned the input and found in a particular column, CITY_NAME contained 15 values that occurred in many rows
  - Those 15 values are each assigned a unique 4-bit code in the column dictionary for the CITY_NAME column for the table
  - DB2 can store the column values for 16 rows into a 64-bit unit of storage
  - When processing that column, DB2 can perform some operations on the 64-bit unit of storage rather than processing each of the 16 column values one at a time

BLU Acceleration Concepts

© Copyright IBM Corporation 2017

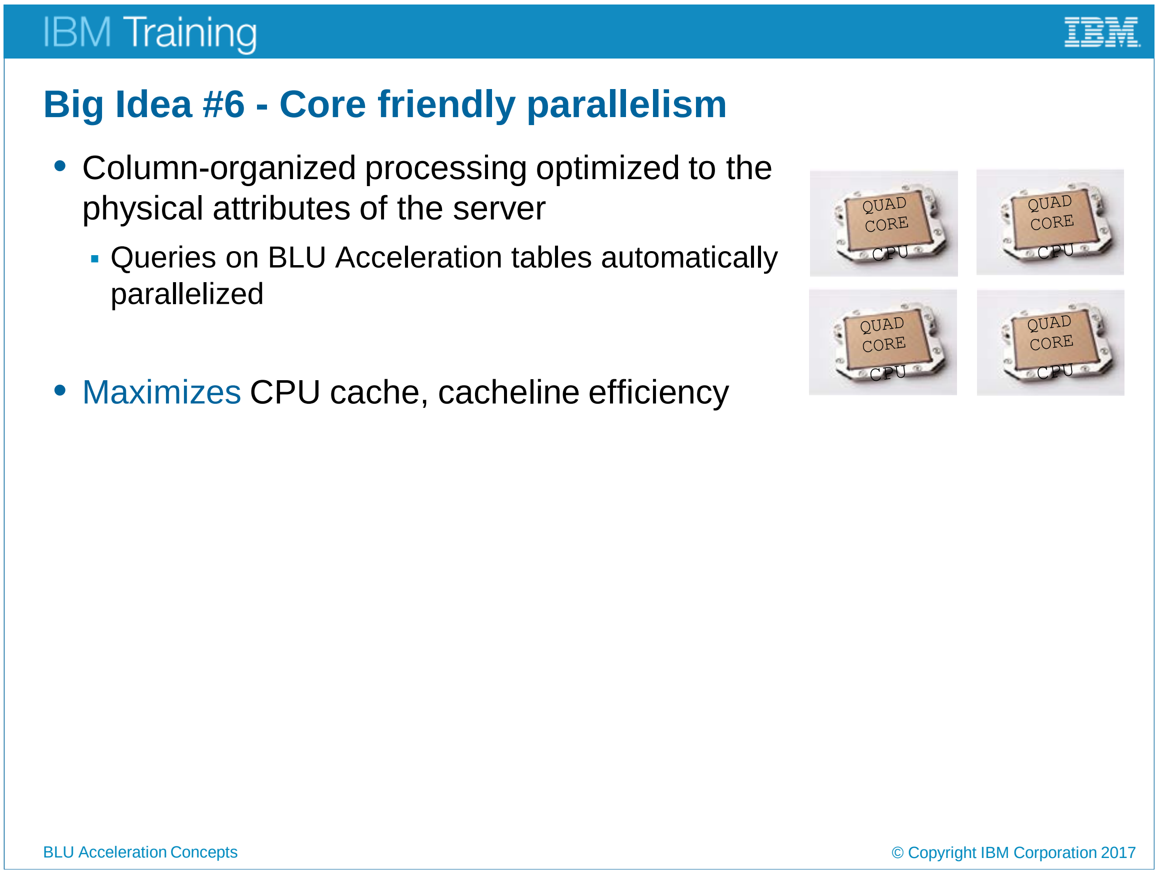
Vector processing engine for processing vectors of column data instead of individual values

One technique used by DB2 to reduce processing costs for queries using column-organized tables is vector processing. This utilizes the encoded data which DB2 stored based on the column based compression dictionaries.

For example, if a table contained a column named CITY_NAME, DB2 might find that fifteen values occurred in many data rows, like ‘NEW YORK’, ‘PARIS’ or ‘TOKYO’. DB2 can assign unique 4-bit codes to each of the fifteen values.

As rows of data are loaded into the table, DB2 creates pages with values from the CITY_NAME column. Any rows with a value equal to the fifteen encoded cities can be stored together into 64-bit units of storage which could hold sixteen values.

Some operations can load the 64-bits of data into a register to be acted on at the same time instead of repeating a single column operation sixteen times.



IBM Training

Big Idea #6 - Core friendly parallelism

- Column-organized processing optimized to the physical attributes of the server
  - Queries on BLU Acceleration tables automatically parallelized
- Maximizes CPU cache, cacheline efficiency

QUAD CORE CPU
QUAD CORE CPU
QUAD CORE CPU
QUAD CORE CPU

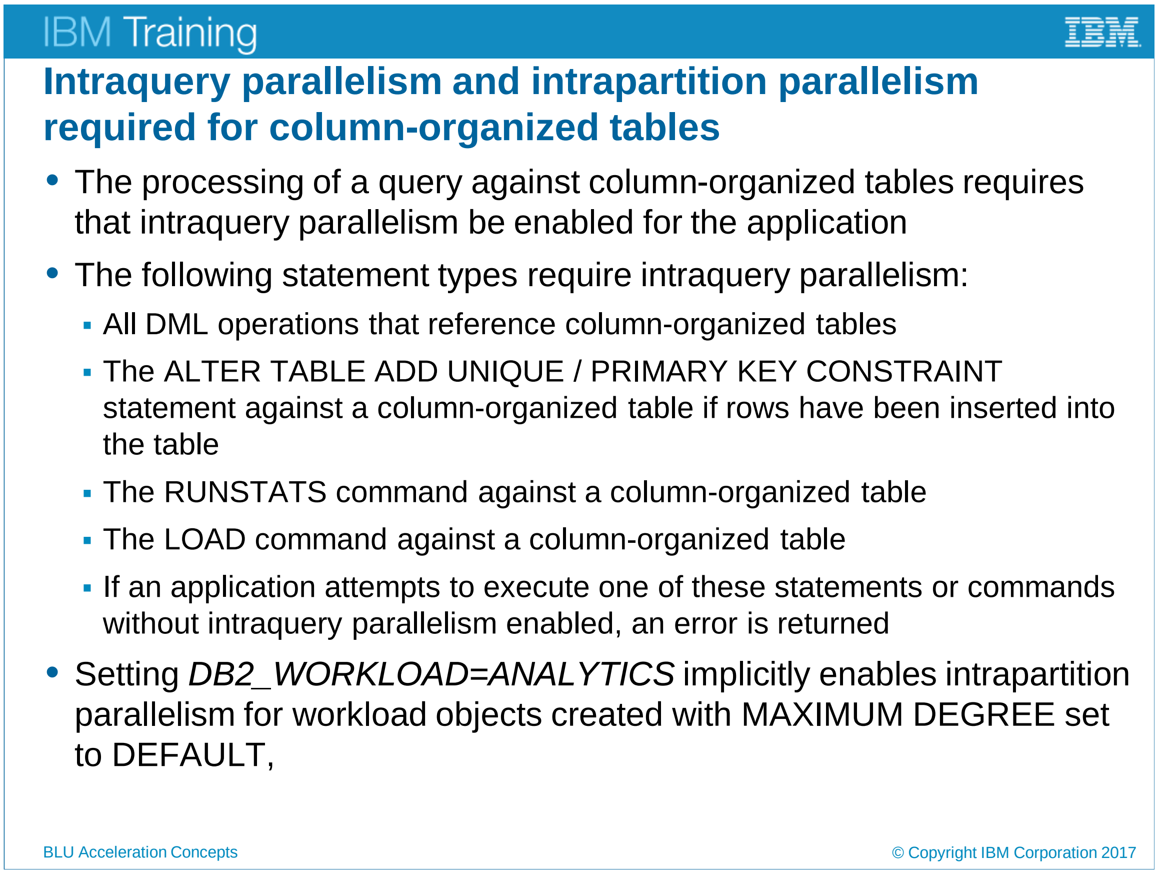
BLU Acceleration Concepts

© Copyright IBM Corporation 2017

Big Idea #6 - Core friendly parallelism

DB2 with BLU Acceleration pays close attention to multi-core parallelism. DB2 with BLU Acceleration is designed from the ground up to take advantage of the cores you have and to always drive multi-core parallelism for the queries you have. This is all done in shared memory - this is not the inter-partition parallelism used for database partitioning.

Part of the code design included processing to optimize efficiency of hardware memory and cache operations to improve performance.



IBM Training

Intraquery parallelism and intrapartition parallelism required for column-organized tables

- The processing of a query against column-organized tables requires that intraquery parallelism be enabled for the application
- The following statement types require intraquery parallelism:
  - All DML operations that reference column-organized tables
  - The ALTER TABLE ADD UNIQUE / PRIMARY KEY CONSTRAINT statement against a column-organized table if rows have been inserted into the table
  - The RUNSTATS command against a column-organized table
  - The LOAD command against a column-organized table
  - If an application attempts to execute one of these statements or commands without intraquery parallelism enabled, an error is returned
- Setting *DB2_WORKLOAD=ANALYTICS* implicitly enables intrapartition parallelism for workload objects created with MAXIMUM DEGREE set to DEFAULT,

BLU Acceleration Concepts © Copyright IBM Corporation 2017

Intraquery parallelism and intrapartition parallelism required for column-organized tables

The processing of a query using column-organized tables requires DB2's intra-query parallelism to be enabled for the application that is compiling and executing the query. The following statement types require intra-query parallelism:

- All DML operations that reference column-organized tables
- The ALTER TABLE ADD UNIQUE / PRIMARY KEY CONSTRAINT statement against a column-organized table if rows have been inserted into the table
- The RUNSTATS command against a column-organized table
- The LOAD command against a column-organized table

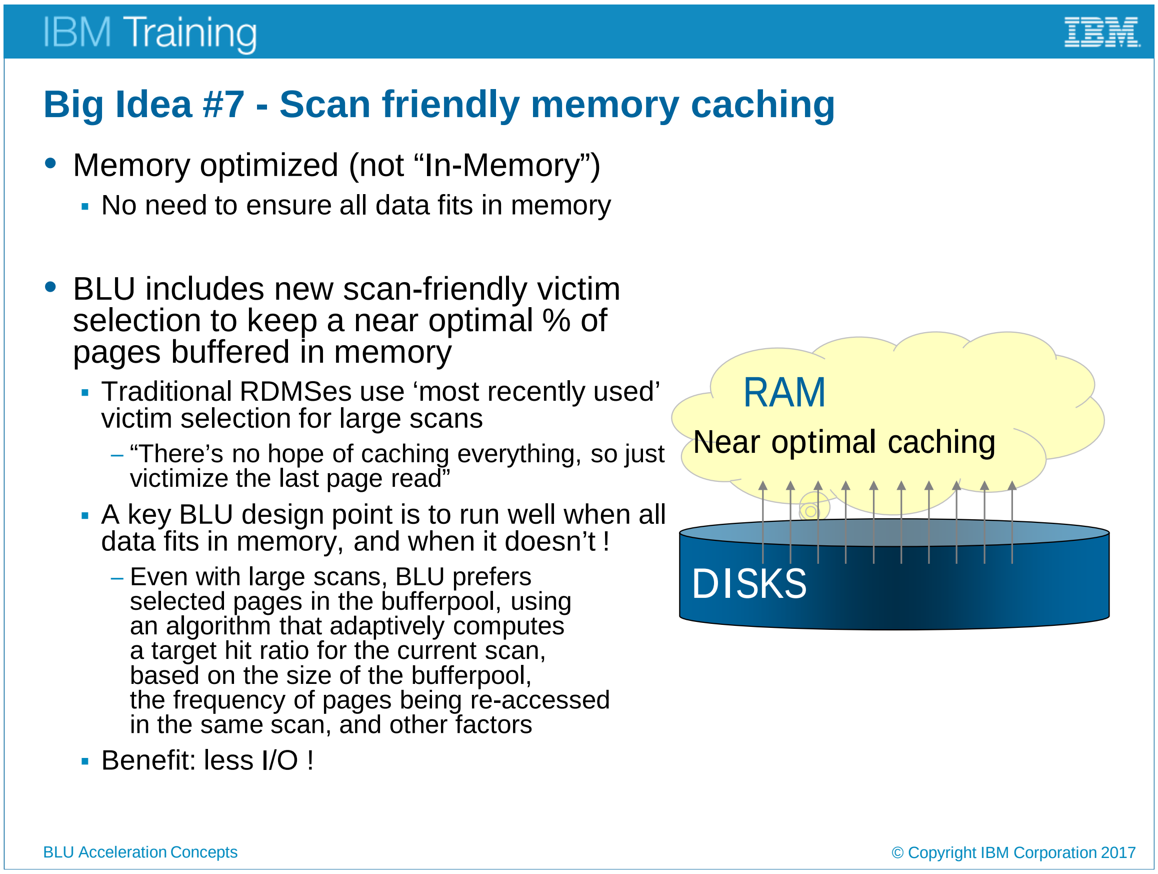
If an application attempts to execute one of these statements or commands without intra-query parallelism enabled, an error is returned.

Access to column-organized tables also requires intrapartition parallelism.

Setting DB2_WORKLOAD=ANALYTICS implicitly enables intrapartition parallelism for workload objects created with MAXIMUM DEGREE set to DEFAULT, and is recommended when using column-organized tables. If this registry variable is not set, or is set to another value, intrapartition parallelism must be explicitly enabled before column-organized tables can be accessed.

Intrapartition parallelism can be enabled or disabled explicitly by using one of the following methods:

- *Instance level*. Intrapartition parallelism can be enabled for the instance by setting the intra_parallel database manager configuration parameter to YES. This setting is not dynamic and requires the instance to be restarted.
- *Database level*. Intrapartition parallelism can be enabled to apply to all applications that map to a particular workload. To enable intrapartition parallelism for a workload, set its MAXIMUM DEGREE attribute to a value greater than 1. This setting is dynamic and will take effect at the beginning of the next unit of work. Users without a specific workload management configuration can set the MAXIMUM DEGREE attribute on SYSDEFAULTUSERWORKLOAD to enable intrapartition parallelism at the database level. Users with a customized workload management configuration can use this attribute to enable intrapartition parallelism for a specific workload, such as a subset of applications that access column-organized tables.
- *Application level*. Intrapartition parallelism can be enabled for an individual application by calling the ADMIN_SET_INTRA_PARALLEL procedure. The setting takes effect at the start of the application's next unit of work. This is a good option if you want intrapartition parallelism disabled at the instance level, but enabled for specific applications that access column-organized tables.


IBM Training

Big Idea #7 - Scan friendly memory caching

- Memory optimized (not "In-Memory")
  - No need to ensure all data fits in memory
- BLU includes new scan-friendly victim selection to keep a near optimal % of pages buffered in memory
  - Traditional RDMSes use 'most recently used' victim selection for large scans
    - "There's no hope of caching everything, so just victimize the last page read"
  - A key BLU design point is to run well when all data fits in memory, and when it doesn't !
    - Even with large scans, BLU prefers selected pages in the bufferpool, using an algorithm that adaptively computes a target hit ratio for the current scan, based on the size of the bufferpool, the frequency of pages being re-accessed in the same scan, and other factors
  - Benefit: less I/O !

BLU Acceleration Concepts © Copyright IBM Corporation 2017

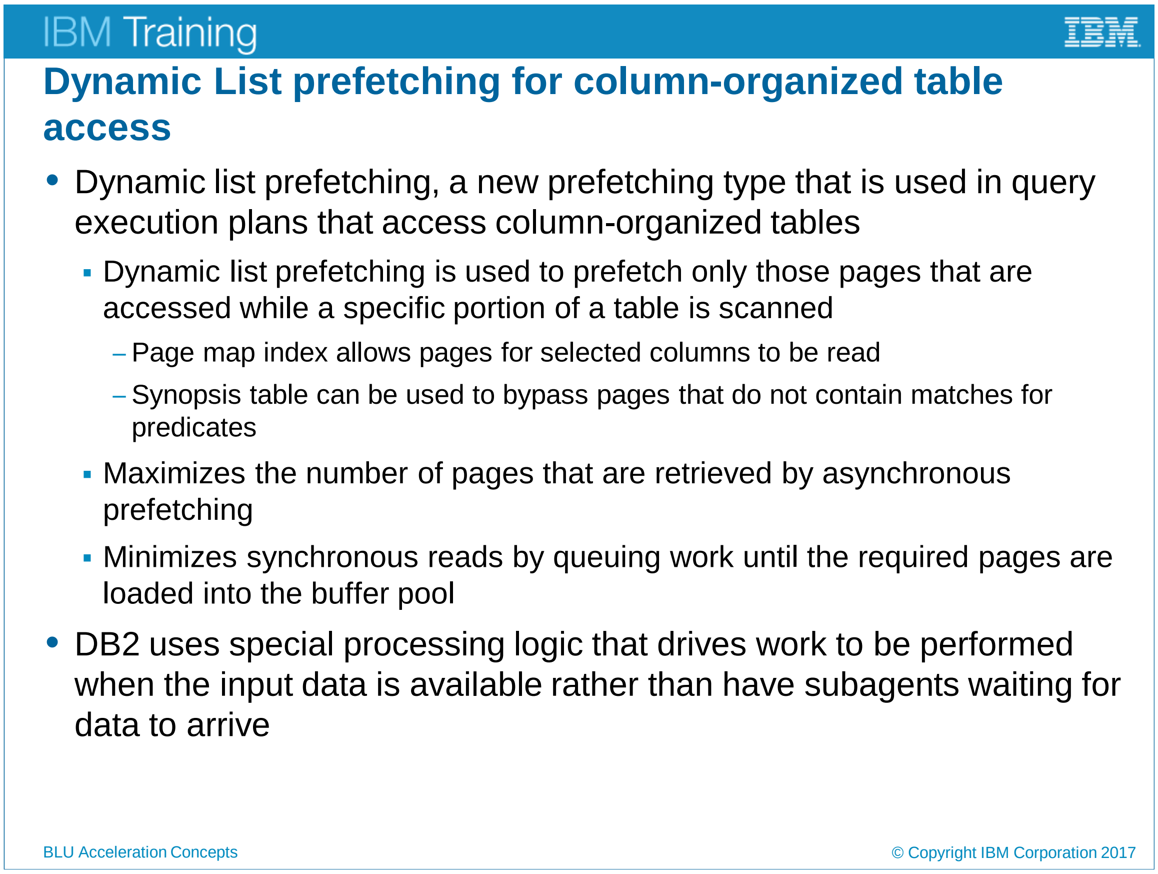
Big Idea #7 - Scan friendly memory caching

DB2 BLU Acceleration column-organized table support includes an enhanced caching strategy for buffer pools to substantially reduce I/O.

With row-organized tables, the need to uncompress the entire row and read the full row of columns into the buffer pool influenced the query processing to perform all of the operations on the columns for that row at the same time.

DB2 uses unique processing logic for column-organized tables that marks buffer pool pages containing column data that will likely need to be accessed again, so they are retained longer in the buffer pool and avoid extra I/Os.

DB2 designed the processing for column-organized tables to fully utilize buffer pool memory but is also optimized to handle larger tables that exceed buffer pool capacity.



IBM Training

Dynamic List prefetching for column-organized table access

- Dynamic list prefetching, a new prefetching type that is used in query execution plans that access column-organized tables
  - Dynamic list prefetching is used to prefetch only those pages that are accessed while a specific portion of a table is scanned
    - Page map index allows pages for selected columns to be read
    - Synopsis table can be used to bypass pages that do not contain matches for predicates
  - Maximizes the number of pages that are retrieved by asynchronous prefetching
  - Minimizes synchronous reads by queuing work until the required pages are loaded into the buffer pool
- DB2 uses special processing logic that drives work to be performed when the input data is available rather than have subagents waiting for data to arrive

BLU Acceleration Concepts © Copyright IBM Corporation 2017

Dynamic List prefetching for column-organized table access

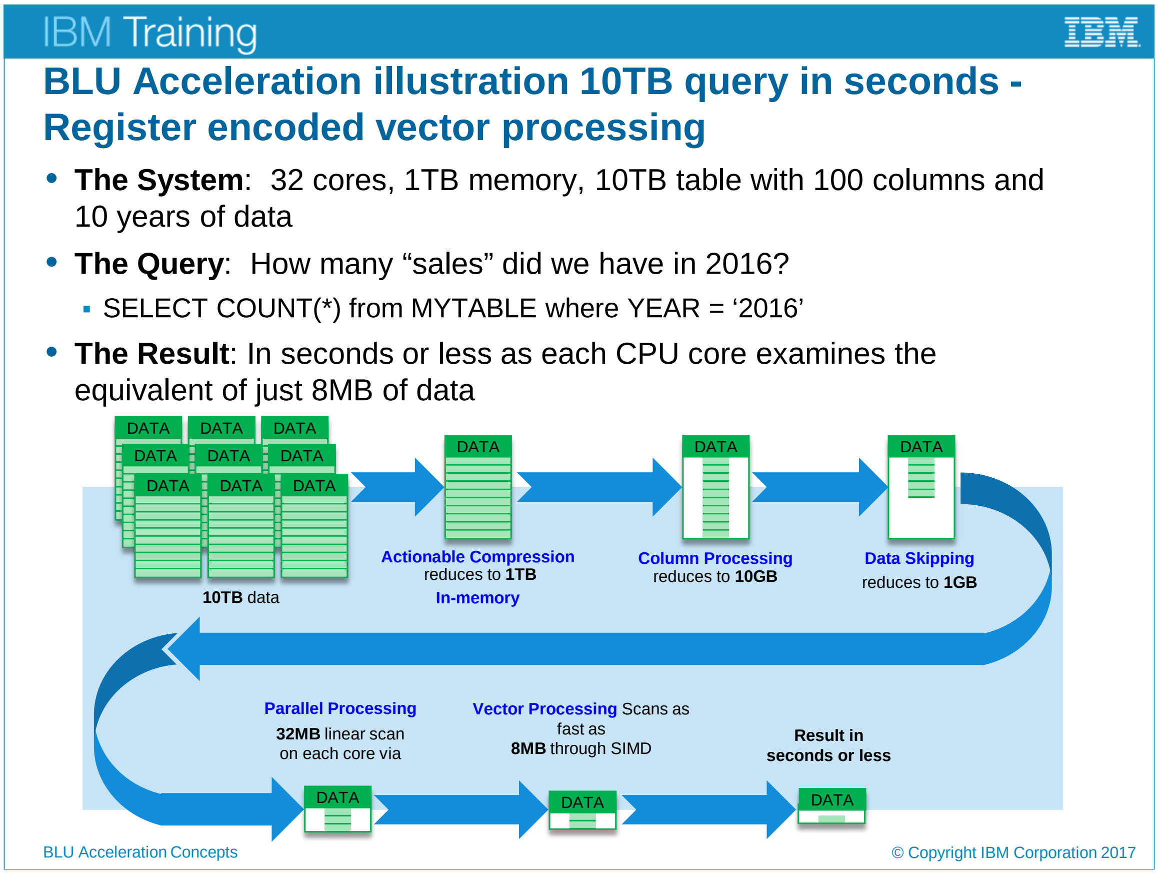
Dynamic list prefetching is used to prefetch exactly those pages that will be accessed while scanning a specific portion of the table.

This prefetch method maximizes the number of pages that are retrieved by asynchronous prefetching (while minimizing synchronous reads) by queuing work until the required pages have been loaded into the buffer pool.

The number of pages that each subagent can prefetch simultaneously is limited by the prefetch size of the table space being accessed (PREFETCHSIZE). The prefetch size can have significant performance implications, particularly for large table scans.

The PREFETCHSIZE clause on either the CREATE TABLESPACE or the ALTER TABLESPACE statement traditionally lets you specify the number of prefetched pages that will be read from the table space when prefetching is performed. The value that you specify (or AUTOMATIC) is stored in the PREFETCHSIZE column of the SYSCAT.TABLESPACES catalog view. Although dynamic list prefetching typically prefetches up to PREFETCHSIZE pages at a time, this might not always be possible and in some cases, the PREFETCHSIZE value might be automatically adjusted for performance reasons. If you observe I/O waits while a query is prefetching data using dynamic list prefetching, try increasing the value of PREFETCHSIZE.

The processing for column-organized tables included logic to drive work to DB2 agents when data is available in the buffer pool to avoid agents performing synchronous waits for data to operate on.



IBM Training

BLU Acceleration illustration 10TB query in seconds - Register encoded vector processing

- **The System**: 32 cores, 1TB memory, 10TB table with 100 columns and 10 years of data
- **The Query**: How many “sales” did we have in 2016?
  - SELECT COUNT(*) from MYTABLE where YEAR = ‘2016’
- **The Result**: In seconds or less as each CPU core examines the equivalent of just 8MB of data

DATA
10TB data
Actionable Compression reduces to 1TB
In-memory
Column Processing reduces to 10GB
Data Skipping reduces to 1GB
Parallel Processing 32MB linear scan on each core via
Vector Processing Scans as fast as 8MB through SIMD
Result in seconds or less

BLU Acceleration Concepts

© Copyright IBM Corporation 2017

BLU Acceleration illustration 10TB query in seconds - Register encoded vector processing

Prior to the implementation of DB2 with BLU Acceleration, the idea of being able to get a query result from a large table with 10 terabytes of data in a second or less without using indexes would seem impossible.

Here’s how the design components of DB2 with BLU acceleration could possibly achieve this.

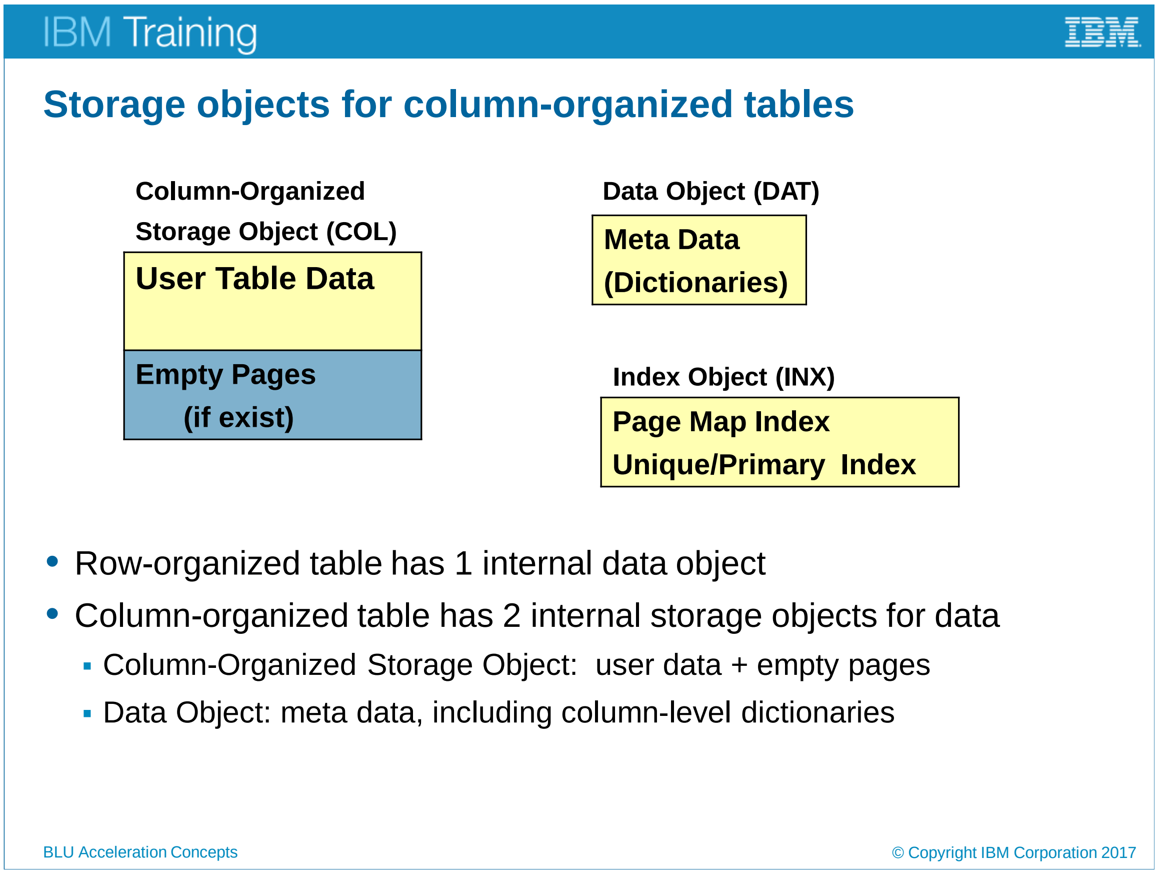
Assume the system has 32 processor cores and 1TB of memory. The table has 10TB of raw application data, 100 columns that hold ten years of information.

A simple query might want to count sales for one year, 2016.

- First, the extreme compression on column-organized tables might reduce the raw 10TB of data into 1TB of storage.
- If the query only accesses one column, then the storage for 99 columns can be bypassed, so the remaining one percent might be 10GB of data.
- Using the synopsis table to perform data skipping we may be able to bypass reading the other 9 years, which in 90 percent, so the query now only needs 1GB of the 10GB for the one column.

- Since the system has 32 CPUs the scan could be divided and processed in parallel which would have 32MB processed by each CPU.
- If the column data is processed using vectors that handle four columns of data per operation, the processing per CPU is now reduced to 8MB of data.

All the processing techniques could work together to reduce the query processing to an amount of processing that could complete quickly.



IBM Training

Storage objects for column-organized tables

- Row-organized table has 1 internal data object
- Column-organized table has 2 internal storage objects for data
  - Column-Organized Storage Object: user data + empty pages
  - Data Object: meta data, including column-level dictionaries

BLU Acceleration Concepts © Copyright IBM Corporation 2017

Storage objects for column-organized tables

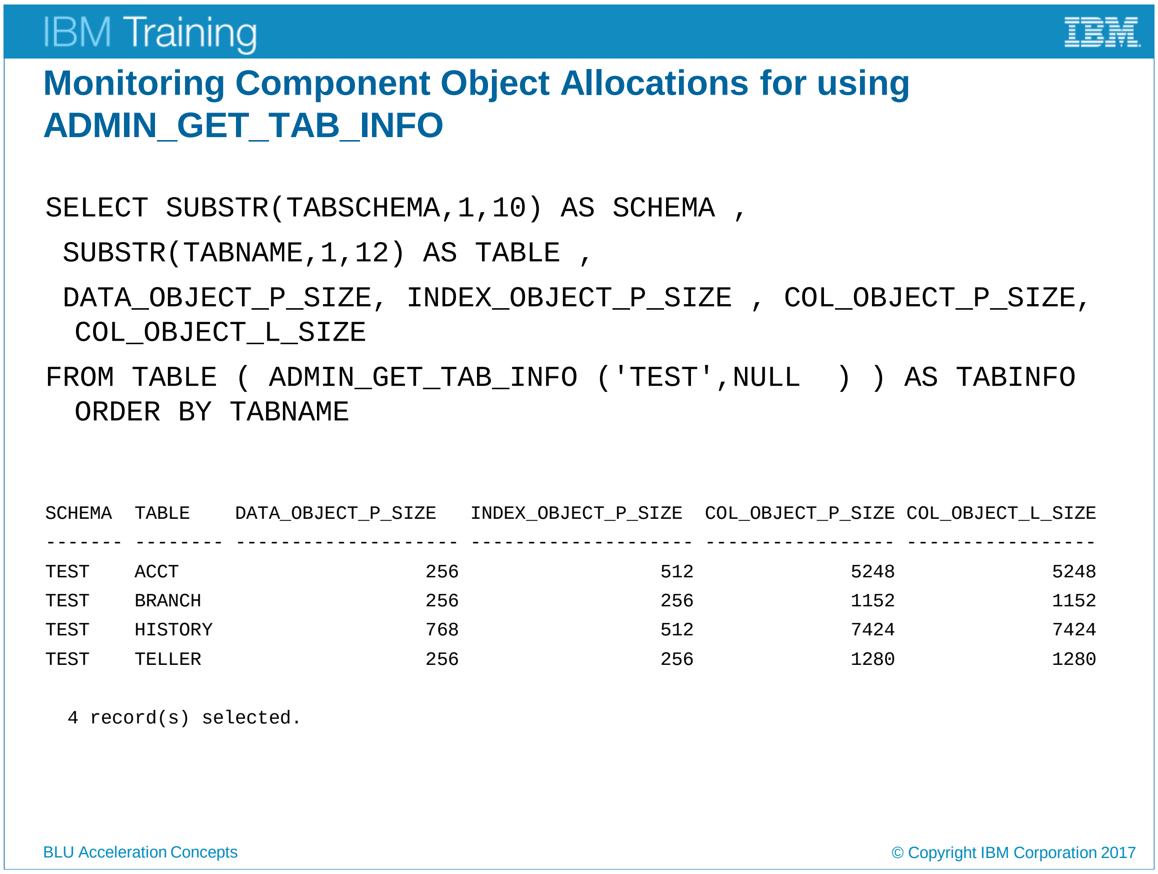
Similar to the implementation of XML data, DB2 implemented a new Column-organized storage object to column-organized tables.

For a row-organized table, the standard type data columns, excluding the LONG and XML data that are not stored inline, are stored in the data object for the table.

For a column-organized table, the user column data is stored in a set of pages termed the column-organized storage object. The column dictionaries and some other table metadata are stored in the data storage object for the table.

The page map index which DB2 creates for each column-organized table will be stored in the index storage object for the table. DB2 will also use the index storage object for any primary key or unique indexes defined on the table.

One major impact of using a new storage object for column organized tables is the reporting of access to the new type of page in DB2 monitoring statistics.


IBM Training

Monitoring Component Object Allocations for using ADMIN_GET_TAB_INFO

```
SELECT SUBSTR(TABSCHEMA,1,10) AS SCHEMA ,
 SUBSTR(TABNAME,1,12) AS TABLE ,
 DATA_OBJECT_P_SIZE, INDEX_OBJECT_P_SIZE , COL_OBJECT_P_SIZE,
  COL_OBJECT_L_SIZE
FROM TABLE ( ADMIN_GET_TAB_INFO ('TEST',NULL  ) ) AS TABINFO
  ORDER BY TABNAME
```

```
SCHEMA  TABLE    DATA_OBJECT_P_SIZE   INDEX_OBJECT_P_SIZE  COL_OBJECT_P_SIZE COL_OBJECT_L_SIZE
------- -------- -------------------- -------------------- ----------------- -----------------
TEST    ACCT                      256                  512              5248              5248
TEST    BRANCH                    256                  256              1152              1152
TEST    HISTORY                   768                  512              7424              7424
TEST    TELLER                    256                  256              1280              1280

  4 record(s) selected.
```

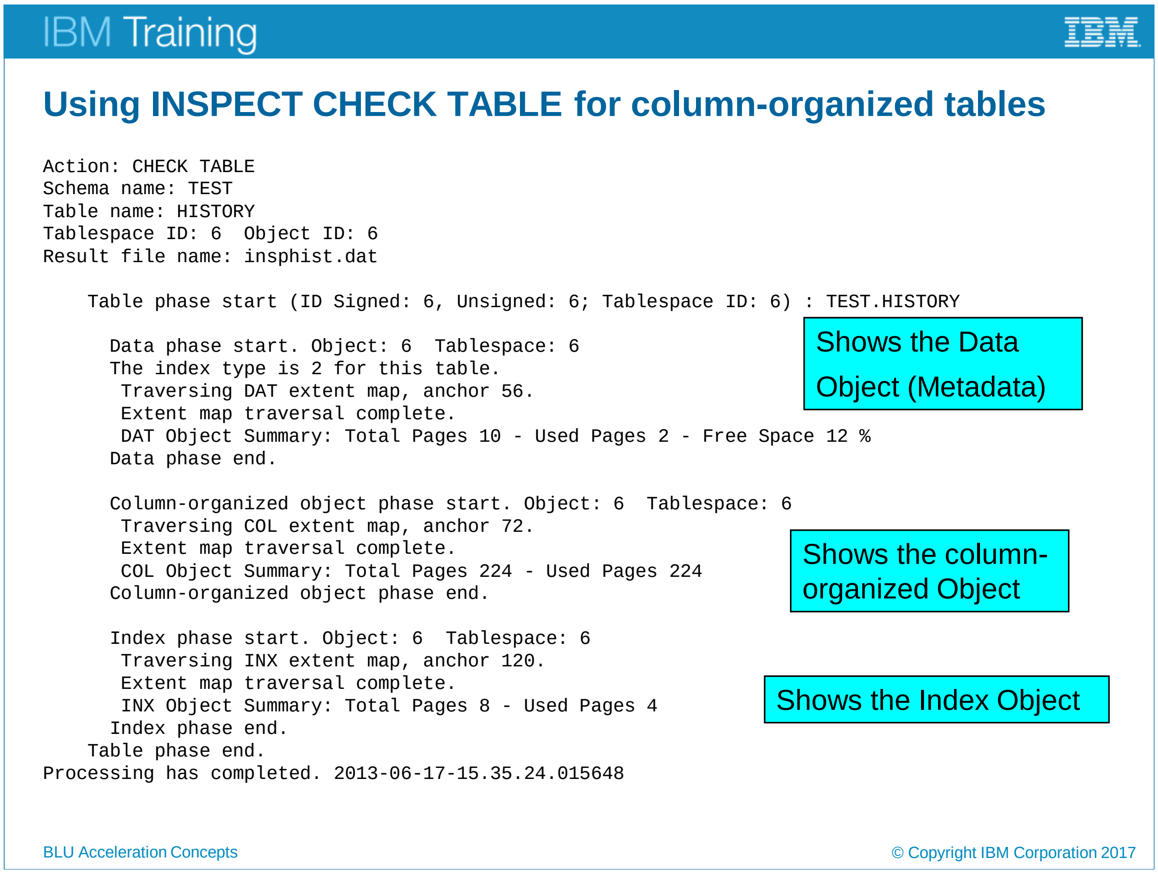
BLU Acceleration Concepts © Copyright IBM Corporation 2017

Monitoring Component Object Allocations for using ADMIN_GET_TAB_INFO

The table function ADMIN_GET_TAB_INFO now includes new columns of information, COL_OBJECT_P_SIZE and COL_OBJECT_L_SIZE that show the storage used for the column organized storage object.

COL_OBJECT_L_SIZE - Amount of disk space logically allocated for the column-organized data in the table, reported in kilobytes.

COL_OBJECT_P_SIZE - Amount of disk space physically allocated for the column-organized data in the table, reported in kilobytes. The size returned takes into account full extents allocated for the table and includes the EMP extents for objects created in DMS table spaces. This size represents the physical size of the base columnar data only



IBM Training

Using INSPECT CHECK TABLE for column-organized tables

```
Action: CHECK TABLE
Schema name: TEST
Table name: HISTORY
Tablespace ID: 6  Object ID: 6
Result file name: insphist.dat

    Table phase start (ID Signed: 6, Unsigned: 6; Tablespace ID: 6) : TEST.HISTORY

      Data phase start. Object: 6  Tablespace: 6
      The index type is 2 for this table.
       Traversing DAT extent map, anchor 56.
       Extent map traversal complete.
       DAT Object Summary: Total Pages 10 - Used Pages 2 - Free Space 12 %
      Data phase end.

      Column-organized object phase start. Object: 6  Tablespace: 6
       Traversing COL extent map, anchor 72.
       Extent map traversal complete.
       COL Object Summary: Total Pages 224 - Used Pages 224
      Column-organized object phase end.

      Index phase start. Object: 6  Tablespace: 6
       Traversing INX extent map, anchor 120.
       Extent map traversal complete.
       INX Object Summary: Total Pages 8 - Used Pages 4
      Index phase end.
    Table phase end.
Processing has completed. 2013-06-17-15.35.24.015648
```

Shows the Data Object (Metadata)

Shows the column-organized Object

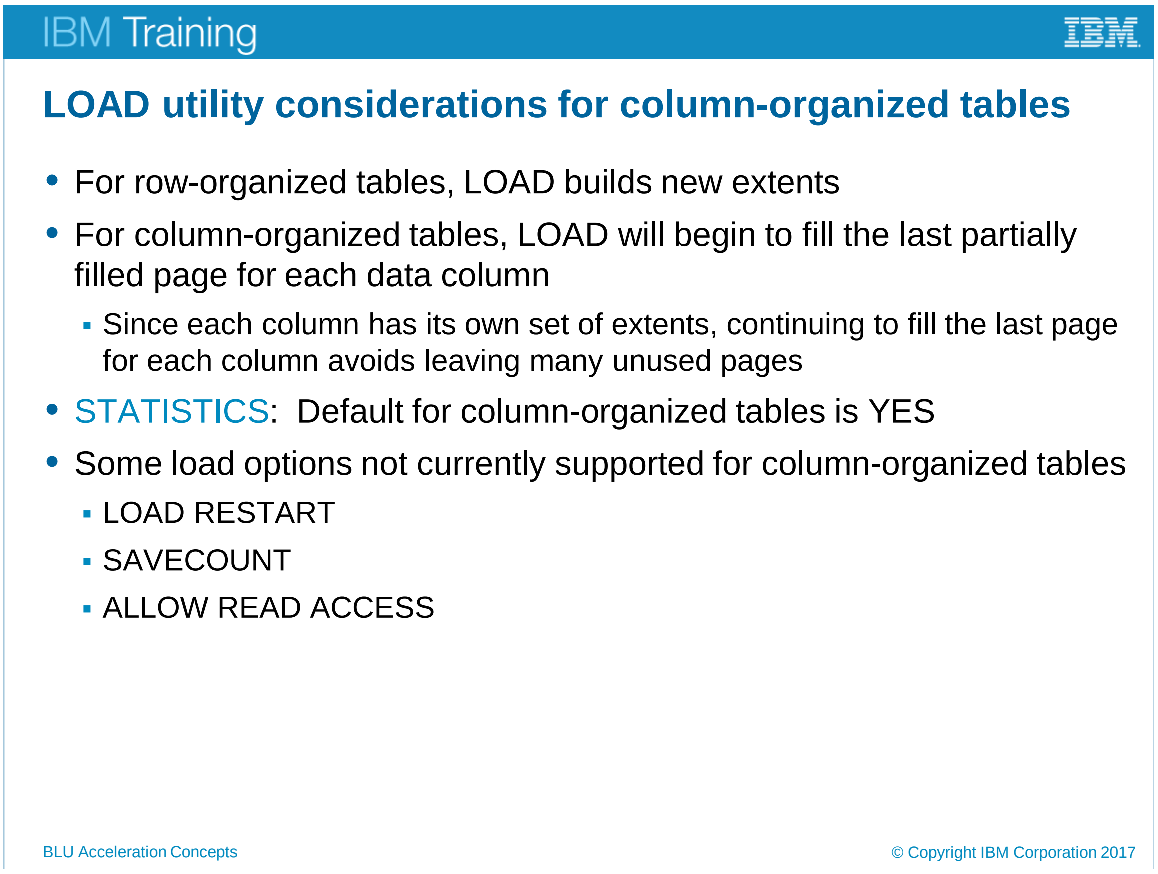
Shows the Index Object

BLU Acceleration Concepts

© Copyright IBM Corporation 2017

Using INSPECT CHECK TABLE for column-organized tables

The slide shows an example of the INSPECT CHECK TABLE report generated using a column-organized table. The report includes the Column-organized object as a new section of the report in addition to the data and index objects. This should be the largest portion of the storage for a column-organized table.


IBM Training

LOAD utility considerations for column-organized tables

- For row-organized tables, LOAD builds new extents
- For column-organized tables, LOAD will begin to fill the last partially filled page for each data column
  - Since each column has its own set of extents, continuing to fill the last page for each column avoids leaving many unused pages
- STATISTICS: Default for column-organized tables is YES
- Some load options not currently supported for column-organized tables
  - LOAD RESTART
  - SAVECOUNT
  - ALLOW READ ACCESS

BLU Acceleration Concepts © Copyright IBM Corporation 2017

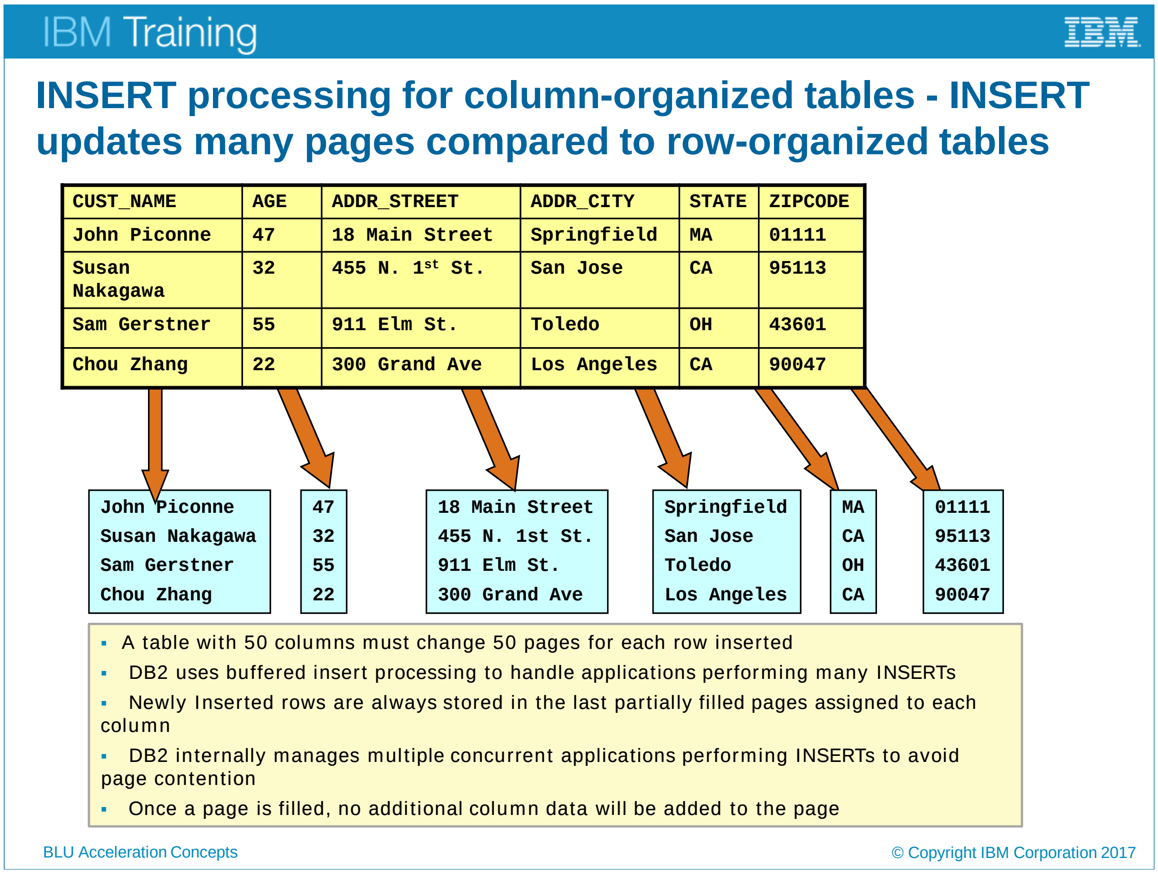
LOAD utility considerations for column-organized tables

For row-organized tables the LOAD utility builds new extents of data pages with the data rows provided as input. If the operation is a LOAD INSERT into a table with existing data, it does not store new data into any pages where data is already stored.

For column-organized tables, when the LOAD utility is used for a LOAD INSERT into a non-empty table, DB2 will find the last partially filled page for each column of the table and continue to fill that page and extent.

For column-organized tables, the default LOAD option is STATISTICS USE PROFILE. A LOAD INSERT operation into a column-organized table maintains table statistics by default if the table was empty at the start of the load operation.

Some LOAD options, including RESTART, SAVECOUNT and ALLOW READ ACCESS are not currently supported for column-organized tables.



IBM Training

INSERT processing for column-organized tables - INSERT updates many pages compared to row-organized tables

CUST_NAME	AGE	ADDR_STREET	ADDR_CITY	STATE	ZIPCODE
John Piconne	47	18 Main Street	Springfield	MA	01111
Susan Nakagawa	32	455 N. 1st St.	San Jose	CA	95113
Sam Gerstner	55	911 Elm St.	Toledo	OH	43601
Chou Zhang	22	300 Grand Ave	Los Angeles	CA	90047

- A table with 50 columns must change 50 pages for each row inserted
- DB2 uses buffered insert processing to handle applications performing many INSERTs
- Newly Inserted rows are always stored in the last partially filled pages assigned to each column
- DB2 internally manages multiple concurrent applications performing INSERTs to avoid page contention
- Once a page is filled, no additional column data will be added to the page

BLU Acceleration Concepts © Copyright IBM Corporation 2017

INSERT processing for column-organized tables - INSERT updates many pages compared to row-organized tables

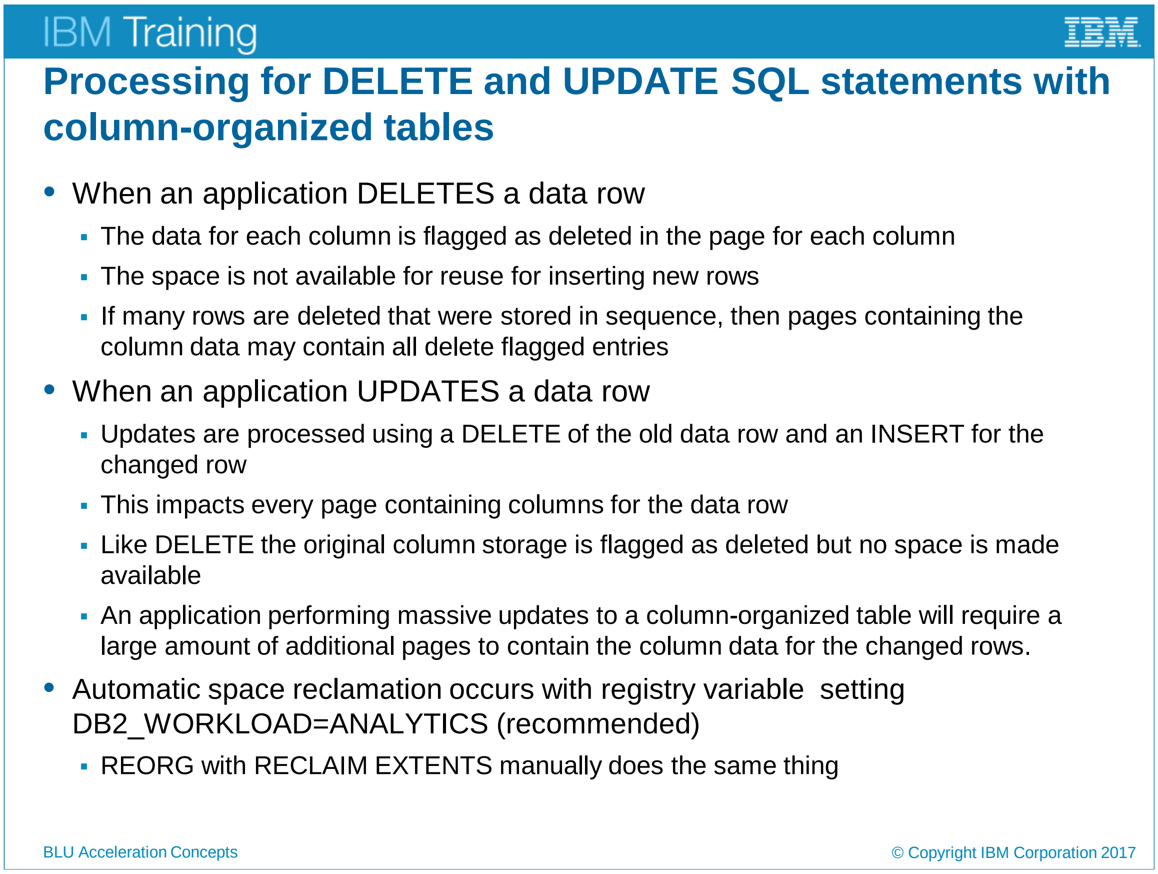
Applications that use SQL INSERT, including DB2 IMPORT and INGEST can run using column-organized tables.

The storage for column-organized tables is optimized for SELECT processing.

Since every column of the column-organized table is stored on a different data page, the processing for inserting a new row of data will need to access and change many pages. For example, inserting one new row in a table with fifty columns will need change the data in fifty pages, which will require logging.

When a row is inserted, the new columns of data will extend the table, using partially filled pages for each column. Once a page for a particular column is filled, no additional column data will be directed to that page, even if other column data is later marked deleted.

DB2 does some special internal processing for inserting rows into column-organized tables that buffers the new data, to reduce the processing overhead when applications are inserting many new rows. That processing also manages the handling for multiple concurrent applications performing inserts into the same table to avoid contention for the pages where new column data will be stored.



IBM Training

Processing for DELETE and UPDATE SQL statements with column-organized tables

- When an application DELETES a data row
  - The data for each column is flagged as deleted in the page for each column
  - The space is not available for reuse for inserting new rows
  - If many rows are deleted that were stored in sequence, then pages containing the column data may contain all delete flagged entries
- When an application UPDATES a data row
  - Updates are processed using a DELETE of the old data row and an INSERT for the changed row
  - This impacts every page containing columns for the data row
  - Like DELETE the original column storage is flagged as deleted but no space is made available
  - An application performing massive updates to a column-organized table will require a large amount of additional pages to contain the column data for the changed rows.
- Automatic space reclamation occurs with registry variable setting DB2_WORKLOAD=ANALYTICS (recommended)
  - REORG with RECLAIM EXTENTS manually does the same thing

BLU Acceleration Concepts © Copyright IBM Corporation 2017

Processing for DELETE and UPDATE SQL statements with column-organized tables

When a row in a column-organized table is deleted, the row is logically (but not physically) deleted. As a result, the space that is used by the deleted row is not available to subsequent transactions, and remains unusable until space reclamation occurs.

For example, consider the case where a table is created and 1 million rows are inserted in batch operation A. The size of the table on disk after batch operation A is 50 MB. After some time, batch operation B inserts another 1 million rows. Now the table consumes 100 MB on disk. At this point, all of the rows that were inserted in batch operation A are deleted, and the table size on disk remains 100 MB. If a third batch operation C inserts another 1 million rows into the table, 50 MB of additional space is required.

With row-organized tables, the rows that are inserted in batch operation C would use the space that was vacated by the deleted rows from batch operation A.

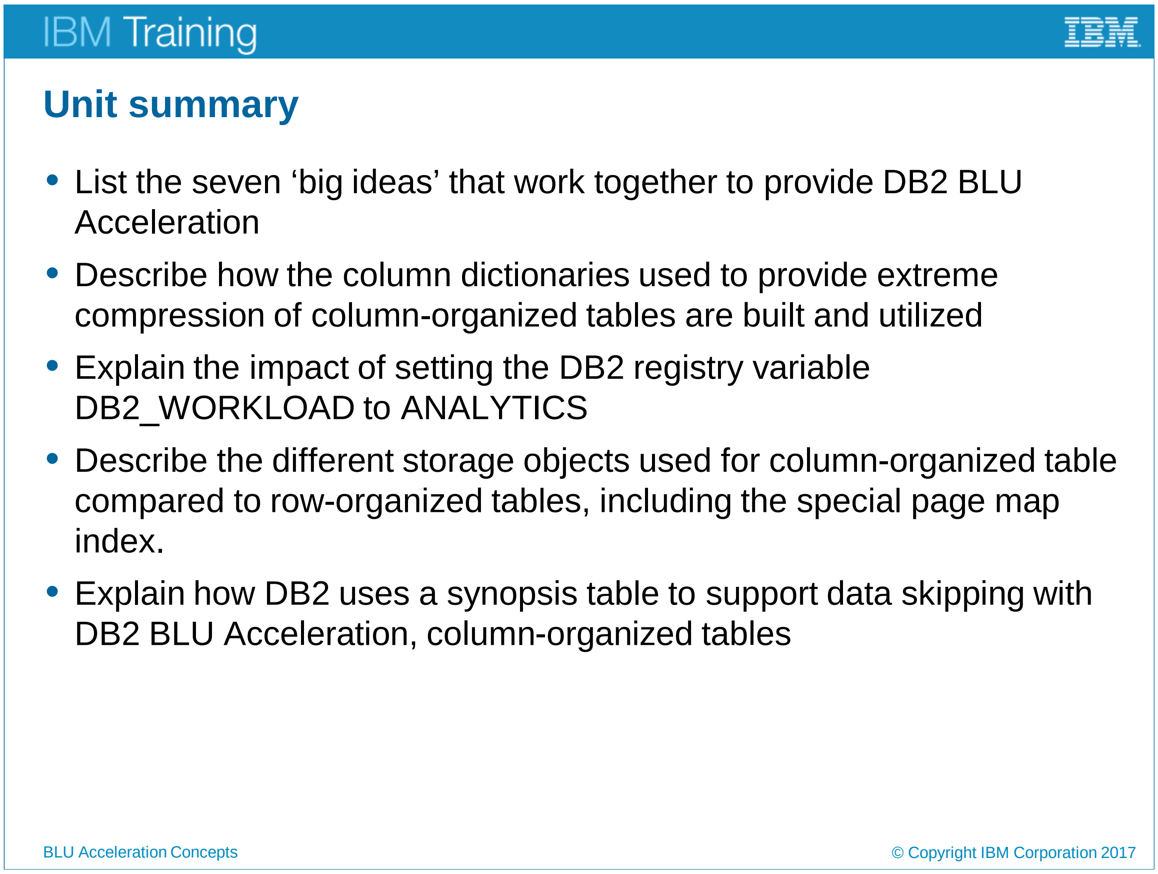
Automatic space reclamation for column organized table occurs when registry variable DB2_WORKLOAD=ANALYTICS is set. This is the recommendation. Alternatively, a REORG TABLE command will reclaim the space that was used by the rows inserted in batch operation A.

When a row in a column-organized table is updated, the row is first deleted and a new copy of the row is inserted at the end of the table.

This means that an updated row consumes space in proportion to the number of times the row has been updated until space reclamation occurs.

All rows in the extent where the update took place must be deleted before any space reclamation will occur.

The number of pages impacted by UPDATE and DELETE processing for column-organized tables would be expected to generate additional logging compared to similar processing for row-organized tables.



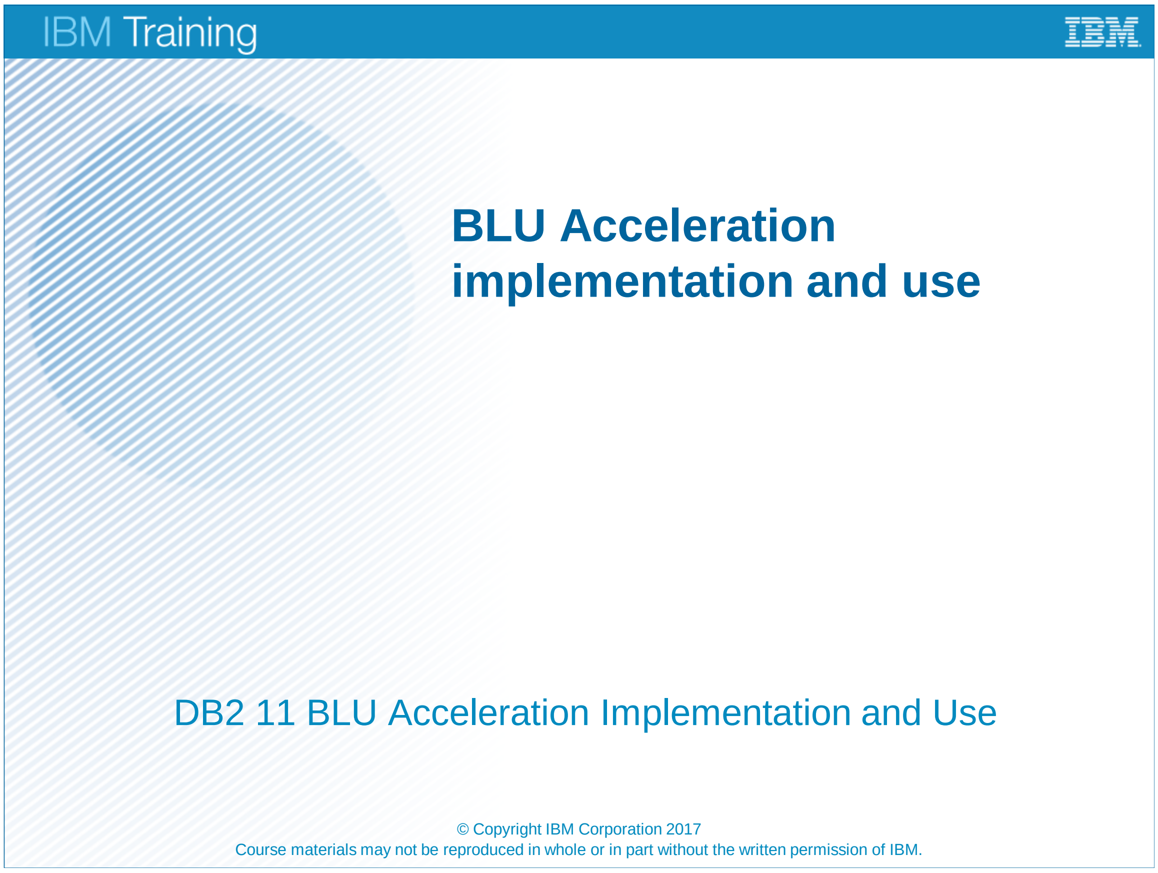
IBM Training

Unit summary

- List the seven ‘big ideas’ that work together to provide DB2 BLU Acceleration
- Describe how the column dictionaries used to provide extreme compression of column-organized tables are built and utilized
- Explain the impact of setting the DB2 registry variable DB2_WORKLOAD to ANALYTICS
- Describe the different storage objects used for column-organized table compared to row-organized tables, including the special page map index.
- Explain how DB2 uses a synopsis table to support data skipping with DB2 BLU Acceleration, column-organized tables

BLU Acceleration Concepts

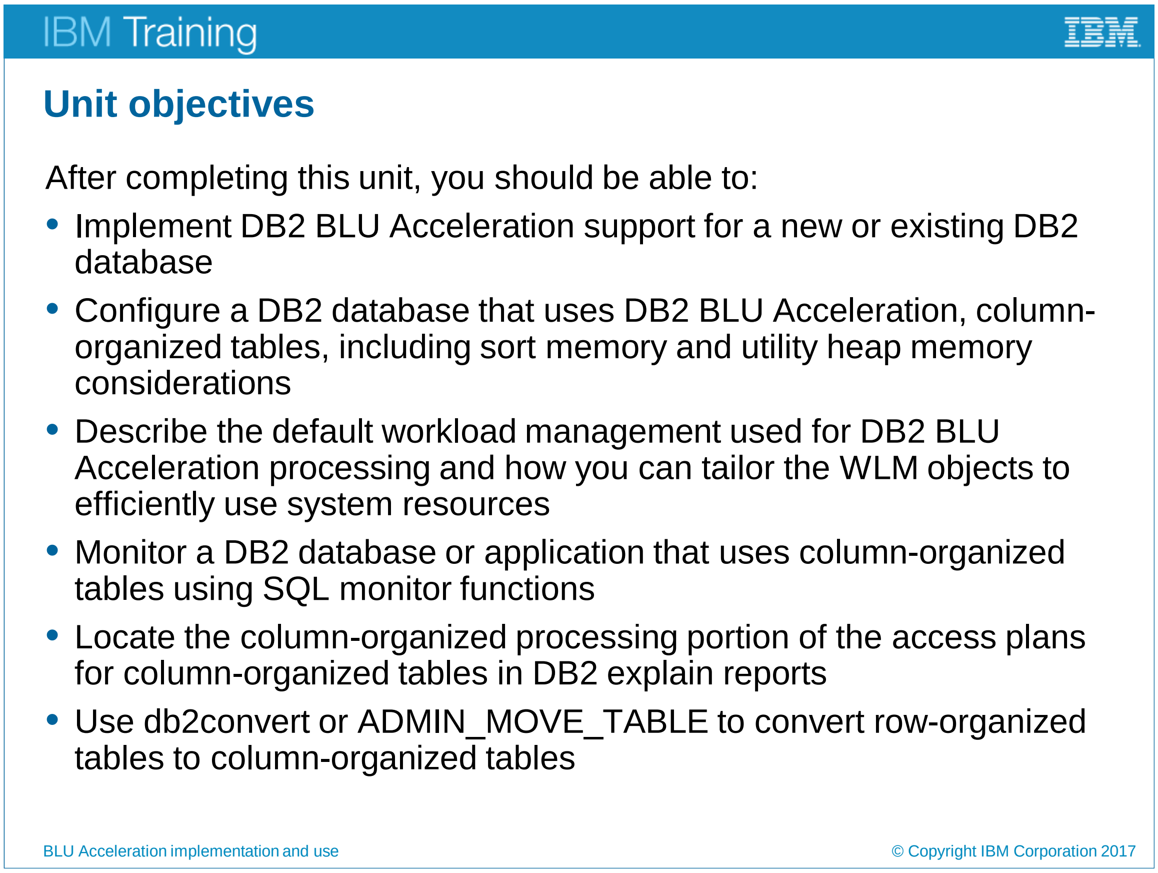
Unit summary



IBM Training

BLU Acceleration implementation and use

DB2 11 BLU Acceleration Implementation and Use



IBM Training

Unit objectives

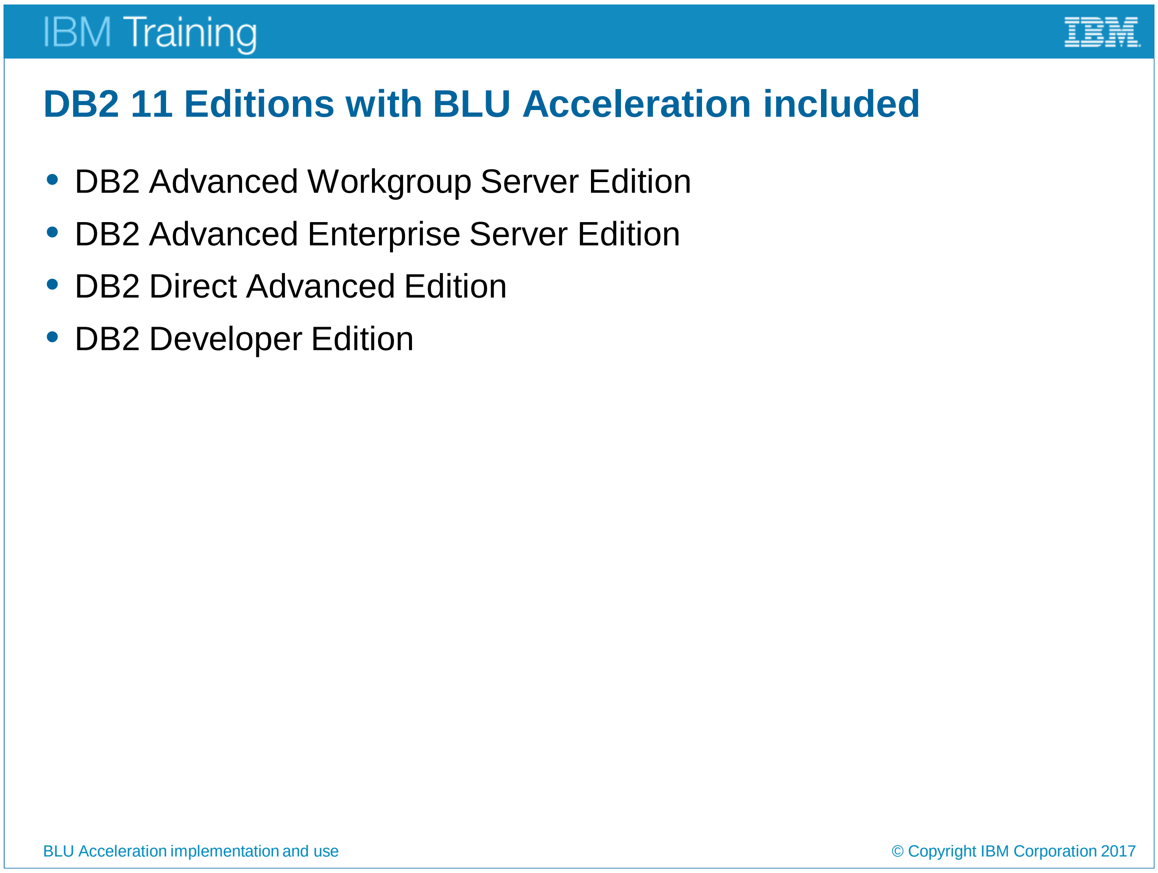
After completing this unit, you should be able to:

- Implement DB2 BLU Acceleration support for a new or existing DB2 database
- Configure a DB2 database that uses DB2 BLU Acceleration, column-organized tables, including sort memory and utility heap memory considerations
- Describe the default workload management used for DB2 BLU Acceleration processing and how you can tailor the WLM objects to efficiently use system resources
- Monitor a DB2 database or application that uses column-organized tables using SQL monitor functions
- Locate the column-organized processing portion of the access plans for column-organized tables in DB2 explain reports
- Use db2convert or ADMIN_MOVE_TABLE to convert row-organized tables to column-organized tables

BLU Acceleration implementation and use

© Copyright IBM Corporation 2017

Unit objectives



IBM Training

DB2 11 Editions with BLU Acceleration included

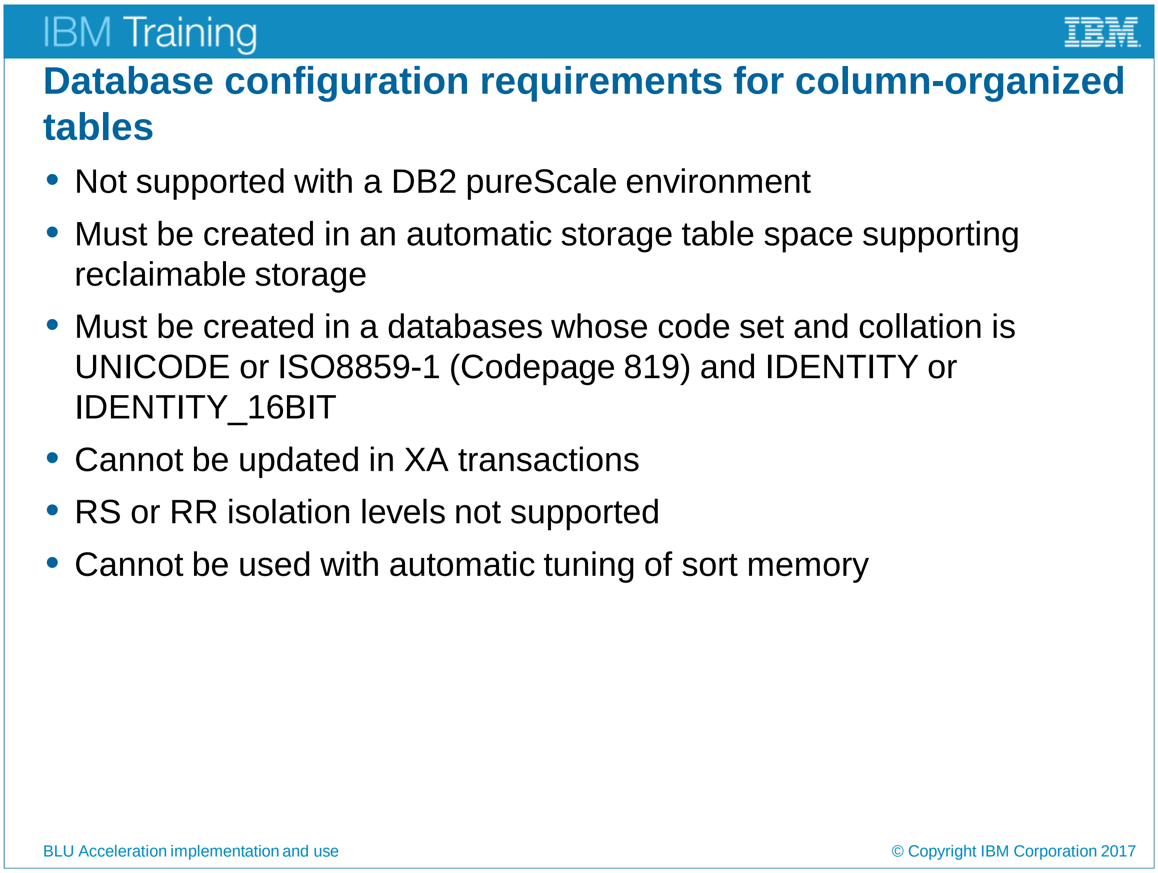
- DB2 Advanced Workgroup Server Edition
- DB2 Advanced Enterprise Server Edition
- DB2 Direct Advanced Edition
- DB2 Developer Edition

BLU Acceleration implementation and use

© Copyright IBM Corporation 2017

DB2 11 Editions with BLU Acceleration included

The slide lists the DB2 11 product editions that include BLU Acceleration feature usage.


IBM Training

Database configuration requirements for column-organized tables

- Not supported with a DB2 pureScale environment
- Must be created in an automatic storage table space supporting reclaimable storage
- Must be created in a databases whose code set and collation is UNICODE or ISO8859-1 (Codepage 819) and IDENTITY or IDENTITY_16BIT
- Cannot be updated in XA transactions
- RS or RR isolation levels not supported
- Cannot be used with automatic tuning of sort memory

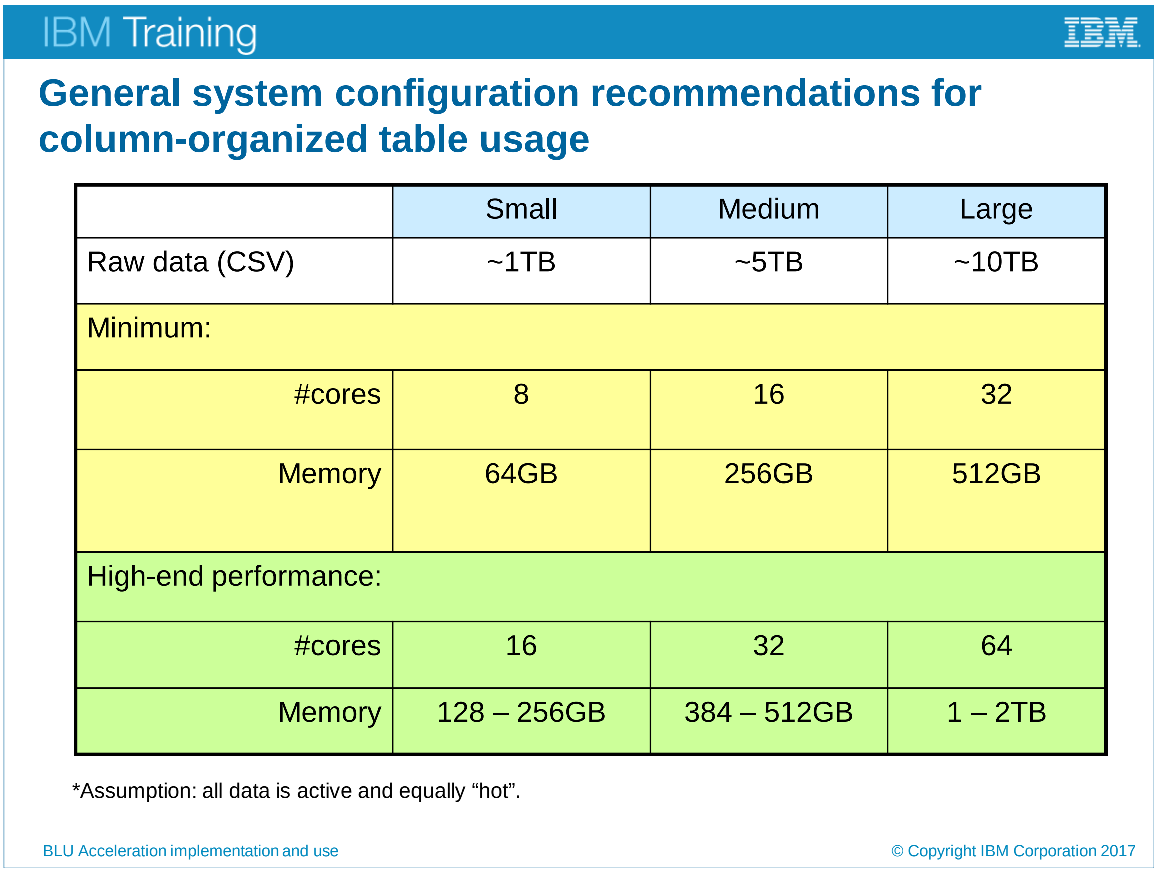
BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Database configuration requirements for column-organized tables

With DB2 11 most database configurations are supported, including massively parallel (MPP) partitioned databases.

Some specific configurations are not currently supported, including DB2 pureScale.

Applications that access column-organized tables cannot utilize lock isolation levels of RS or RR.



IBM Training

General system configuration recommendations for column-organized table usage

	Small	Medium	Large
Raw data (CSV)	~1TB	~5TB	~10TB
Minimum:			
#cores	8	16	32
Memory	64GB	256GB	512GB
High-end performance:			
#cores	16	32	64
Memory	128 – 256GB	384 – 512GB	1 – 2TB

*Assumption: all data is active and equally "hot".

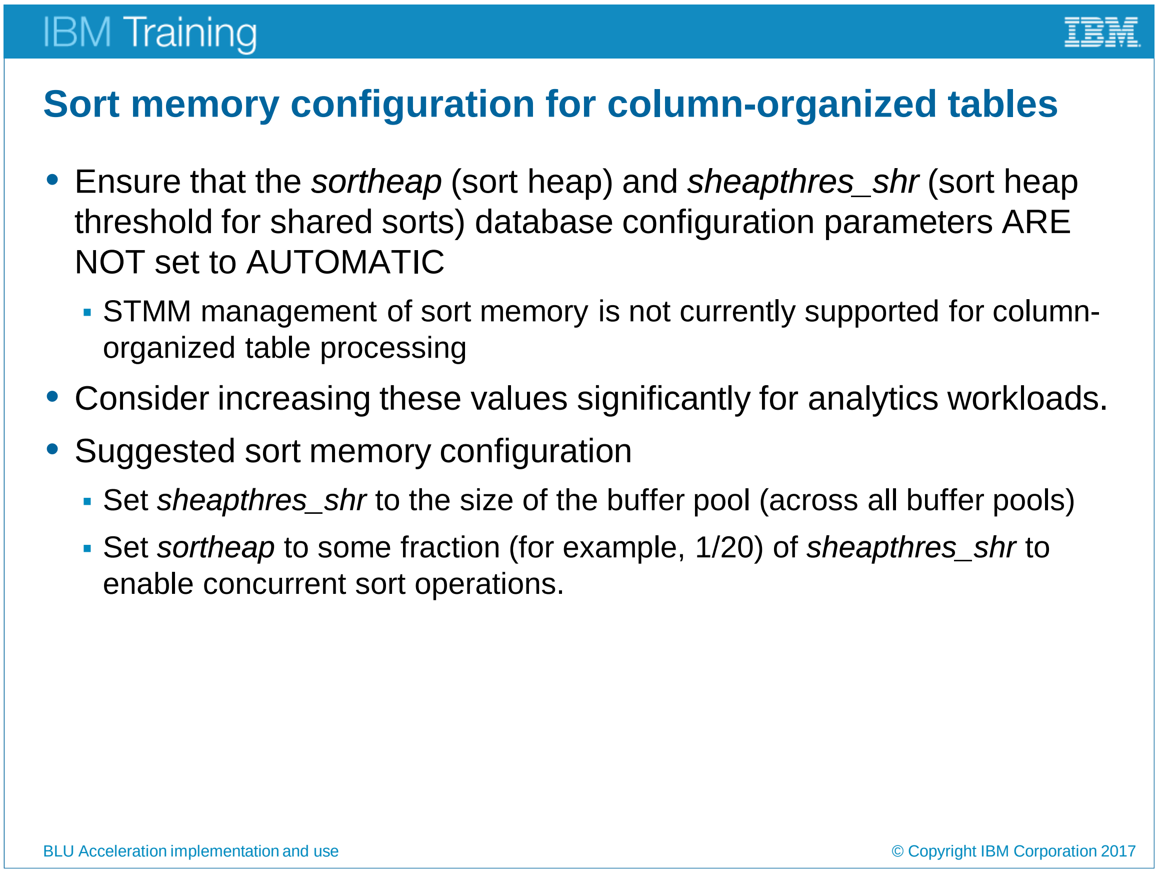
BLU Acceleration implementation and use

© Copyright IBM Corporation 2017

General system configuration recommendations for column-organized table usage

The table shows recommended minimum and 'high-end' performance configurations based on the amount of row application data stored in the column-organized tables. One assumption is that all of the table data will be accessed frequently.

In general the system processor and memory configurations are based on tests with column-organized tables. The DB2 BLU Acceleration routines were designed for systems with large system memory and multiple processors.



IBM Training

Sort memory configuration for column-organized tables

- Ensure that the *sortheap* (sort heap) and *sheapthres_shr* (sort heap threshold for shared sorts) database configuration parameters ARE NOT set to AUTOMATIC
  - STMM management of sort memory is not currently supported for column-organized table processing
- Consider increasing these values significantly for analytics workloads.
- Suggested sort memory configuration
  - Set *sheapthres_shr* to the size of the buffer pool (across all buffer pools)
  - Set *sortheap* to some fraction (for example, 1/20) of *sheapthres_shr* to enable concurrent sort operations.

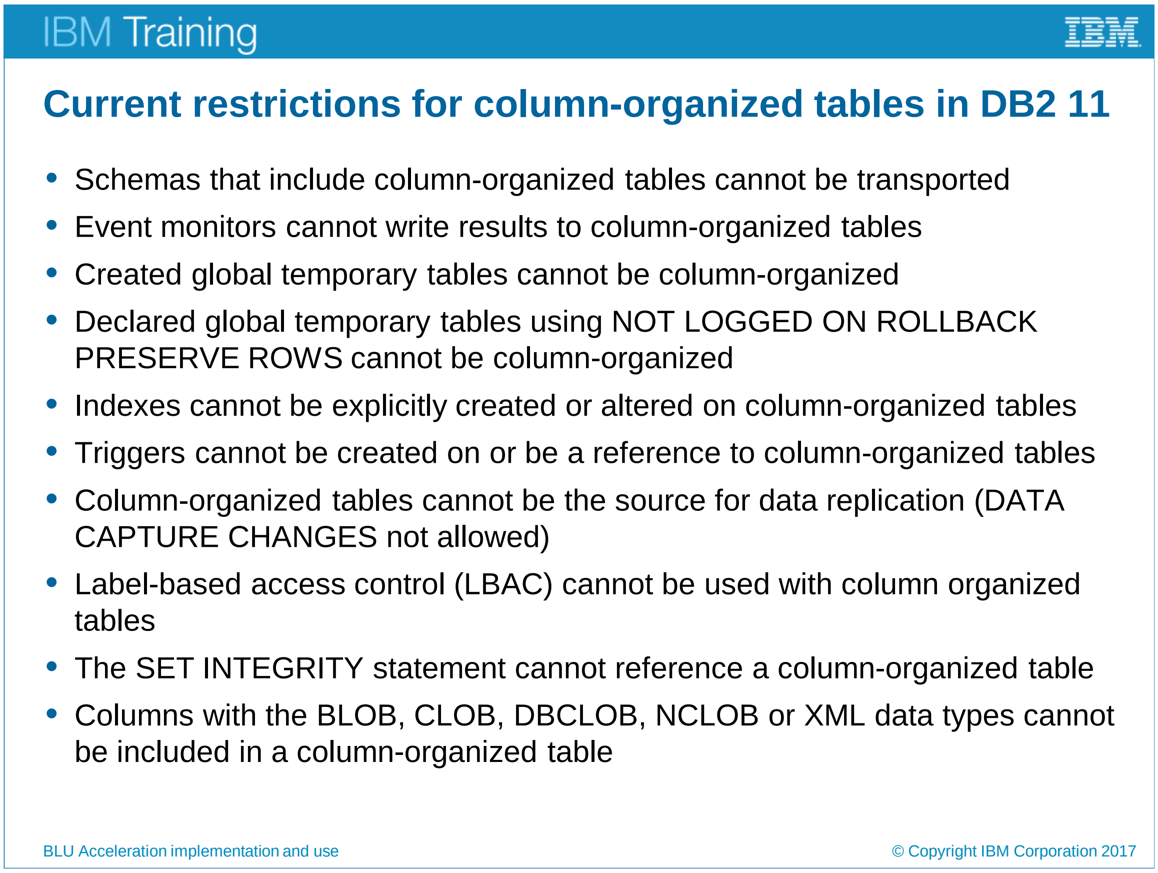
BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Sort memory configuration for column-organized tables

The processing for column-organized tables makes heavy use of database shared sort memory. When column-organized tables are joined, either nested loop joins or hash joins can be performed by the columnar engine. The hash join uses sort memory for holding the data from the inner table of the join. Other join techniques can only be performed by DB2 internally first converting the data to a row format.

DB2 does not current support the use of the self tuning memory management of the sort memory options, sortheap (sort heap) and sheapthres_shr (sort heap threshold for shared sorts, with column-organized tables. These database configuration parameters cannot be set to AUTOMATIC.

Consider increasing these values significantly for analytics workloads. A reasonable starting point is setting sheapthres_shr to the size of the buffer pool (across all buffer pools). Set sortheap to some fraction (for example, 1/20) of sheapthres_shr to enable concurrent sort operations.



IBM Training

Current restrictions for column-organized tables in DB2 11

- Schemas that include column-organized tables cannot be transported
- Event monitors cannot write results to column-organized tables
- Created global temporary tables cannot be column-organized
- Declared global temporary tables using NOT LOGGED ON ROLLBACK PRESERVE ROWS cannot be column-organized
- Indexes cannot be explicitly created or altered on column-organized tables
- Triggers cannot be created on or be a reference to column-organized tables
- Column-organized tables cannot be the source for data replication (DATA CAPTURE CHANGES not allowed)
- Label-based access control (LBAC) cannot be used with column organized tables
- The SET INTEGRITY statement cannot reference a column-organized table
- Columns with the BLOB, CLOB, DBCLOB, NCLOB or XML data types cannot be included in a column-organized table

BLU Acceleration implementation and use

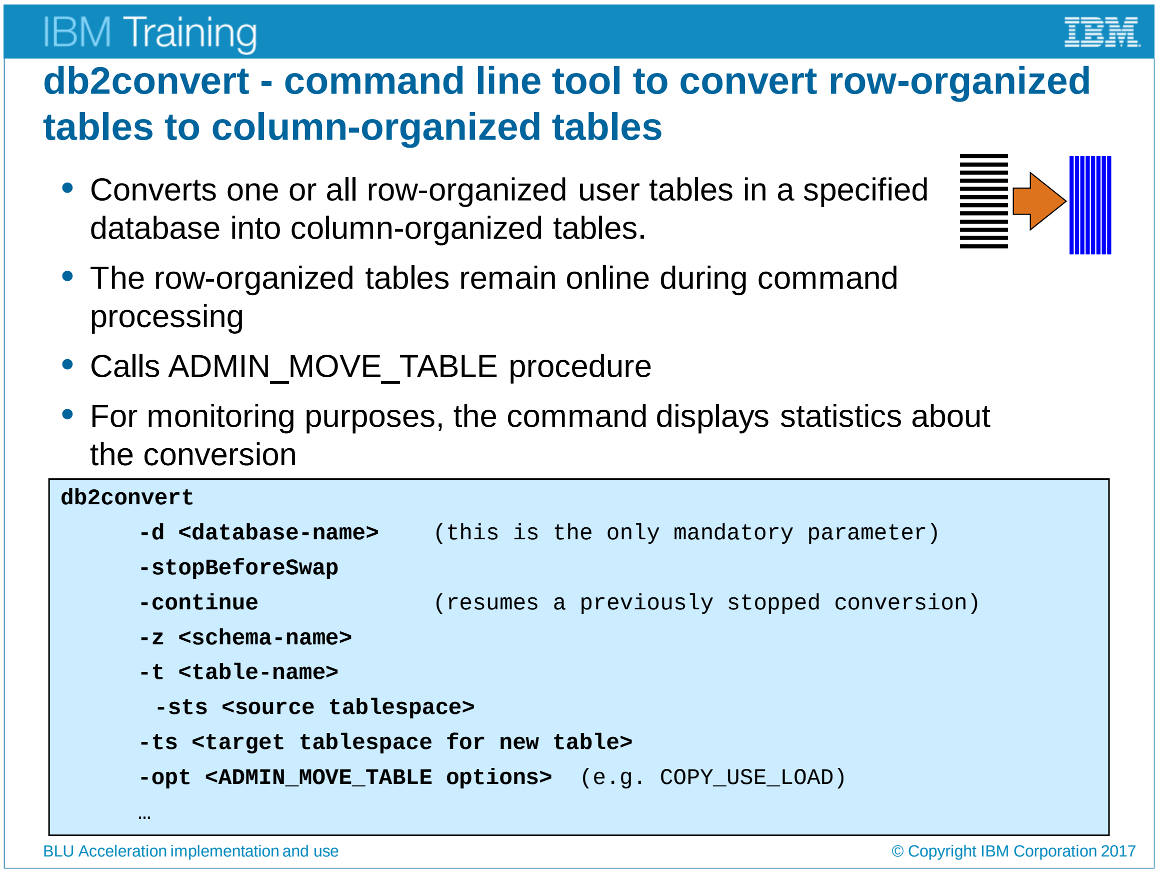
© Copyright IBM Corporation 2017

Current restrictions for column-organized tables in DB2 11

The following additional restrictions apply to column-organized tables in the DB2 Version 11 release:

- Schemas that include column-organized tables cannot be transported
- Event monitors cannot write results to column-organized tables
- Created global temporary tables cannot be column-organized
- Declared global temporary tables using NOT LOGGED ON ROLLBACK PRESERVE ROWS cannot be column-organized
- Indexes cannot be explicitly created or altered on column-organized tables
- Triggers cannot be created on or be a reference to column-organized tables
- Column-organized tables cannot be the source for data replication (DATA CAPTURE CHANGES not allowed)
- Label-based access control (LBAC) cannot be used with column organized tables
- The SET INTEGRITY statement cannot reference a column-organized table
- The REORG TABLE command cannot reference a column-organized table

- Positioned-delete and update statements are not supported for column-organized tables
- The following restrictions apply to a column-organized MQT:
  - MQT's other than shadow tables must reference tables with the same organization as the MQT
  - The ORGANIZE BY COLUMN must be specified when creating a column-organized MQT, even if the dft_table_org database configuration parameter is set to COLUMN
- For a column-organized MQT, the following tables are supported:
  - Shadow tables
  - User-maintained MQTs
  - System-maintained MQTs that are defined with the REFRESH DEFERRED and DISTRIBUTE BY REPLICATION clauses
- A column-organized table cannot be a:
  - range-partitioned table
  - multi-dimensional clustered table
  - temporal table
  - typed table
- Columns with the BLOB, CLOB, DBCLOB, NCLOB or XML data types cannot be included in a column-organized table
- Columns in a column-organized table cannot be dropped or altered
- Enforced check and foreign key constraints are not supported by column-organized tables
- The ROW CHANGE TIMESTAMP generated column option cannot be specified for columns in a column-organized table



IBM Training

db2convert - command line tool to convert row-organized tables to column-organized tables

- Converts one or all row-organized user tables in a specified database into column-organized tables.
- The row-organized tables remain online during command processing
- Calls ADMIN_MOVE_TABLE procedure
- For monitoring purposes, the command displays statistics about the conversion

```
db2convert
      -d <database-name>    (this is the only mandatory parameter)
      -stopBeforeSwap
      -continue             (resumes a previously stopped conversion)
      -z <schema-name>
      -t <table-name>
       -sts <source tablespace>
      -ts <target tablespace for new table>
      -opt <ADMIN_MOVE_TABLE options>  (e.g. COPY_USE_LOAD)
      …
```

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

db2convert - command line tool to convert row-organized tables to column-organized tables

You can use the db2convert command to convert one or all row-organized user tables into column-organized tables in a specified database.

The row-organized tables remain online during command processing. The command displays statistics about the conversion for monitoring purposes.

You must have SQLADM or DBADM authority to invoke the ADMIN_MOVE_TABLE stored procedure, on which the db2convert command depends. You must also have the appropriate object creation authorities, including the authority to issue the SELECT statement on the source table.

The syntax for the db2convert command tool is as follows:

```
>>-db2convert-- -d--database_name-------------------------------->
>--+---------------------------------+--+-------------+------>

   +- -cancel--------------------------+  '- -u--creator-'
   '-+- -stopBeforeSwap-+--+---------+-'
     '- -continue-------'  '- -check-'

>--+--------------------------------+-------------------------->
   '- -z--schema--+----------------+-'
                  '- -t--table_name-'

>--+--------------------------------------------------------+-->
   +- -ts--target_tablespace_name-----------------------------+
   '- -dts--data_tablespace_name-- -its--index_tablespace_name-'

>--+-----------------------------+---------------------------->
   '- -sts--source_tablespace_name-'

>--+-----------------------+--+--------+------------------->
   |        .-COPY_USE_LOAD-. |  '- -trace-'
   '- -opt--+-AMT_options---+-'

>--+-----------------------------+--+--------+-------------->
   '- -usr--userid-- -pw--password-'  '- -force-'

>--+---------------------+----------------------------------><
   '- -o--output_file_name-'
```

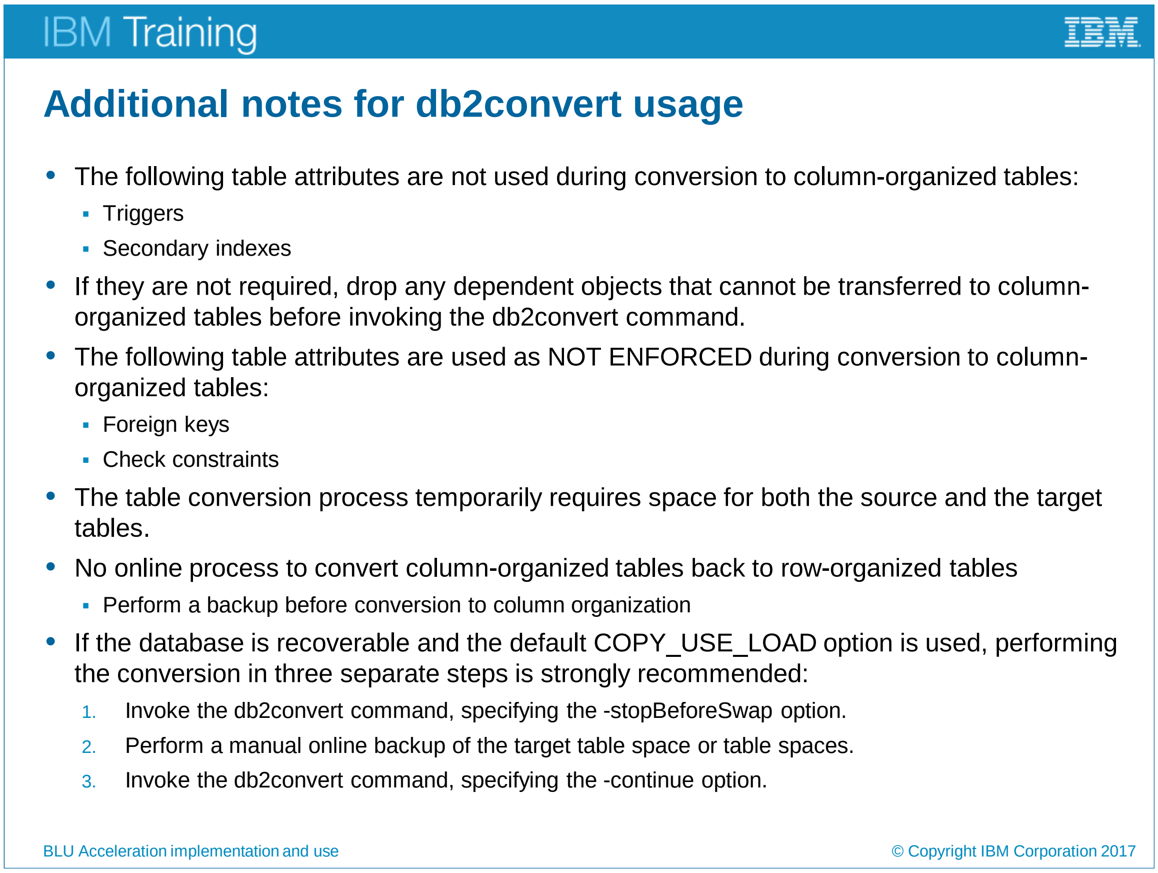
You cannot convert the following table types into column-organized tables:

- Range clustered tables
- Typed tables
- Materialized query tables
- Declared global temporary tables
- Created global temporary tables

Range partitioned tables, MDC tables, and ITC tables are not converted by default. To convert these table types, use the -force option.

Example: To convert all of the tables in MYDATABASE into column-organized tables, issue the following command (after performing a full database backup operation):

```
db2convert -d mydatabase
```



IBM Training

Additional notes for db2convert usage

- The following table attributes are not used during conversion to column-organized tables:
  - Triggers
  - Secondary indexes
- If they are not required, drop any dependent objects that cannot be transferred to column-organized tables before invoking the db2convert command.
- The following table attributes are used as NOT ENFORCED during conversion to column-organized tables:
  - Foreign keys
  - Check constraints
- The table conversion process temporarily requires space for both the source and the target tables.
- No online process to convert column-organized tables back to row-organized tables
  - Perform a backup before conversion to column organization
- If the database is recoverable and the default COPY_USE_LOAD option is used, performing the conversion in three separate steps is strongly recommended:
  1. Invoke the db2convert command, specifying the -stopBeforeSwap option.
  2. Perform a manual online backup of the target table space or table spaces.
  3. Invoke the db2convert command, specifying the -continue option.

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Additional notes for db2convert usage

Here are some additional notes regarding using db2convert to convert row-organized tables to column-organized tables.

The following table attributes are not used when converting to column-organized tables:

- Triggers
- Secondary indexes

If they are not required, drop any dependent objects that cannot be transferred to column-organized tables before invoking the **db2convert** command.

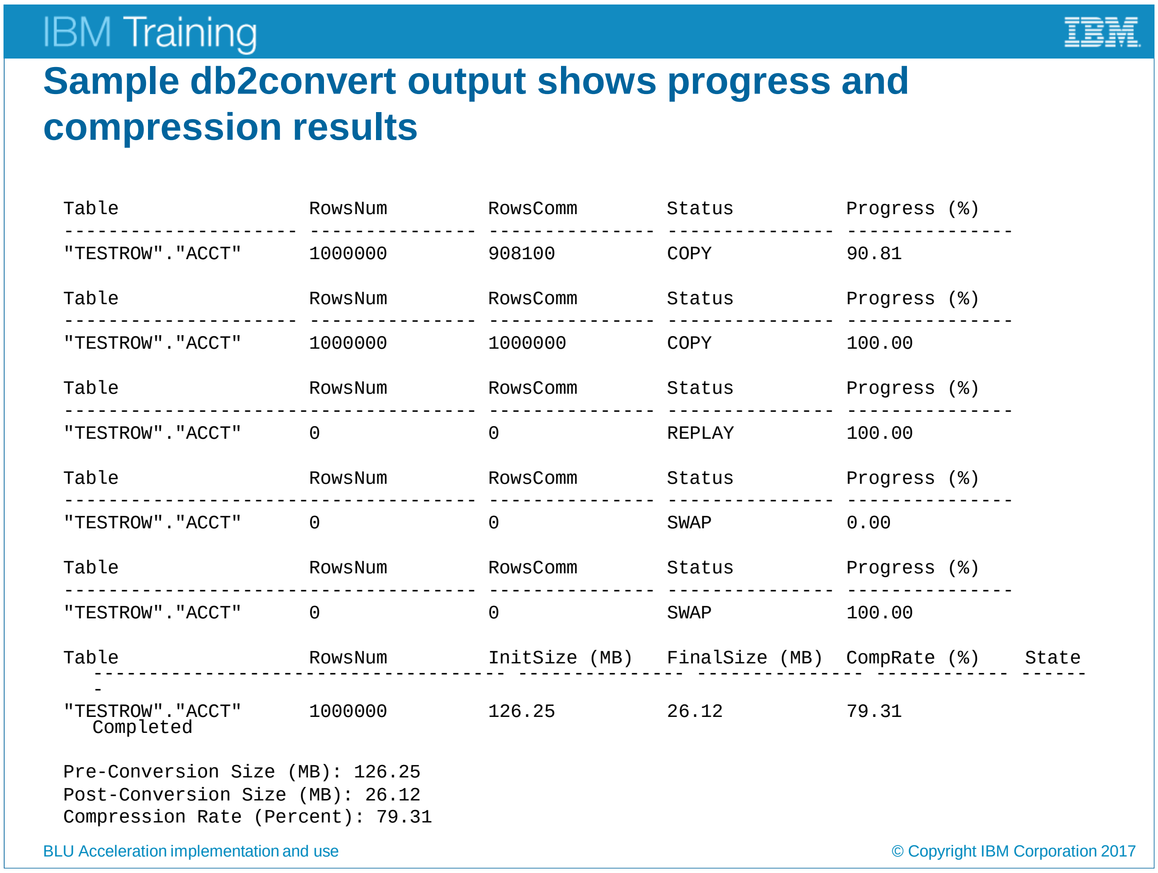
The following table attributes are used as NOT ENFORCED when converting to column-organized tables:

- Foreign keys
- Check constraints

The table conversion process temporarily requires space for both the source and the target table. Because there is no online process to convert column-organized tables back to row-organized tables, the best practice is to perform a backup before converting the tables to column organization.

If the database is recoverable and you don't specify -opt parameter, or if using the -opt parameter, you don't specify COPY_USE_LOAD with sub-option COPY YES, performing the conversion in three separate steps is strongly recommended in order to ensure recoverability:

1. Invoke the db2convert command, specifying the -stopBeforeSwap option.
2. Perform a manual online backup of the target table space or table spaces.
3. Invoke the db2convert command, specifying the -continue option.



IBM Training

Sample db2convert output shows progress and compression results

```
Table                 RowsNum         RowsComm        Status          Progress (%)
--------------------- --------------- --------------- --------------- ---------------
"TESTROW"."ACCT"      1000000         908100          COPY            90.81

Table                 RowsNum         RowsComm        Status          Progress (%)
--------------------- --------------- --------------- --------------- ---------------
"TESTROW"."ACCT"      1000000         1000000         COPY            100.00

Table                 RowsNum         RowsComm        Status          Progress (%)
------------------------------------- --------------- --------------- ---------------
"TESTROW"."ACCT"      0               0               REPLAY          100.00

Table                 RowsNum         RowsComm        Status          Progress (%)
------------------------------------- --------------- --------------- ---------------
"TESTROW"."ACCT"      0               0               SWAP            0.00

Table                 RowsNum         RowsComm        Status          Progress (%)
------------------------------------- --------------- --------------- ---------------
"TESTROW"."ACCT"      0               0               SWAP            100.00

Table                 RowsNum         InitSize (MB)   FinalSize (MB)  CompRate (%)    State
   ------------------------------------- --------------- --------------- ------------- ------
   -
"TESTROW"."ACCT"      1000000         126.25          26.12           79.31
   Completed

Pre-Conversion Size (MB): 126.25
Post-Conversion Size (MB): 26.12
Compression Rate (Percent): 79.31
```

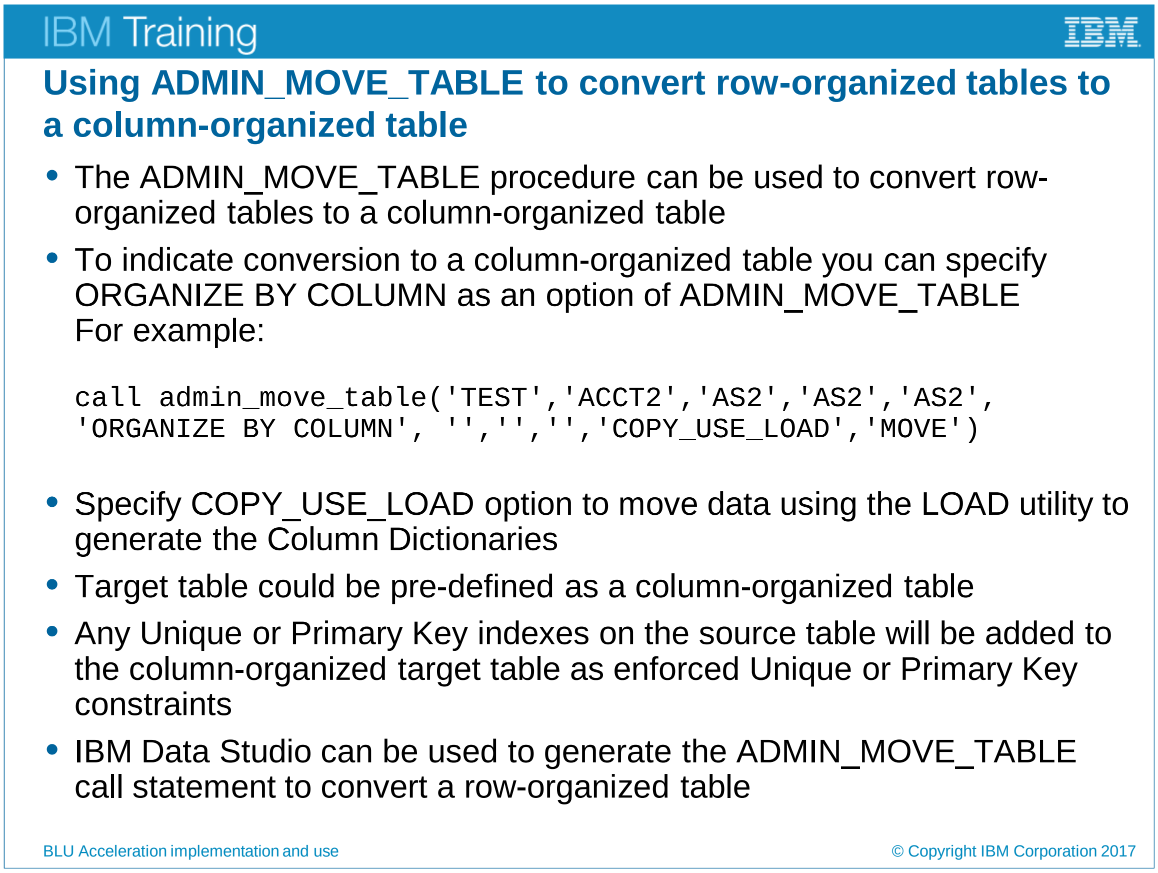
BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Sample db2convert output shows progress and compression results

The slide shows a portion of the output generated from a db2convert command that was run to convert a single table from row-organization to column-organization.

The db2convert command calls the ADMIN_MOVE_TABLE procedure to perform the table conversion. The output shows progress and completion of each phase of processing.

There is also a summary that estimates the saving in storage space following the table conversion.



IBM Training

Using ADMIN_MOVE_TABLE to convert row-organized tables to a column-organized table

- The ADMIN_MOVE_TABLE procedure can be used to convert row-organized tables to a column-organized table
- To indicate conversion to a column-organized table you can specify ORGANIZE BY COLUMN as an option of ADMIN_MOVE_TABLE
  For example:

```
call admin_move_table('TEST','ACCT2','AS2','AS2','AS2',
'ORGANIZE BY COLUMN', '','','','COPY_USE_LOAD','MOVE')
```

- Specify COPY_USE_LOAD option to move data using the LOAD utility to generate the Column Dictionaries
- Target table could be pre-defined as a column-organized table
- Any Unique or Primary Key indexes on the source table will be added to the column-organized target table as enforced Unique or Primary Key constraints
- IBM Data Studio can be used to generate the ADMIN_MOVE_TABLE call statement to convert a row-organized table

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Using ADMIN_MOVE_TABLE to convert row-organized tables to a column-organized table

Using ADMIN_MOVE_TABLE to convert row-organized tables into column-organized tables

Conversion can be achieved in either of the following two ways:

- By specifying a column-organized target table
- By specifying the ORGANIZE BY COLUMN clause as the organize_by_clause parameter.

The ADMIN_MOVE_TABLE stored procedure keeps the row table online during the move operation. When a row-organized table is being moved into a column-organized table, applicable column-organized table restrictions on queries (that is, limited isolation levels) start at the end of processing, after the new table becomes visible to queries.

ADMIN_MOVE_TABLE requires triggers on the source table to capture changes. Because triggers are currently not supported on column-organized tables, the source table cannot be a column-organized table (SQL2103N).

Indexes on column-organized tables are not supported. ADMIN_MOVE_TABLE silently converts primary key and unique indexes into primary key or unique constraints and ignores all non-unique indexes.

You cannot use the ADMIN_MOVE_TABLE procedure to convert a row-organized table into a column-organized table if the table contains unique indexes that are defined on nullable columns.

ADMIN_MOVE_TABLE provides two call parameter options. In one mode, the procedure call parameters define the changes to be made and the procedure creates the target table.

For example to use the ADMIN_MOVE_TABLE stored procedure to convert the row-organized STAFF table into a column-organized table.

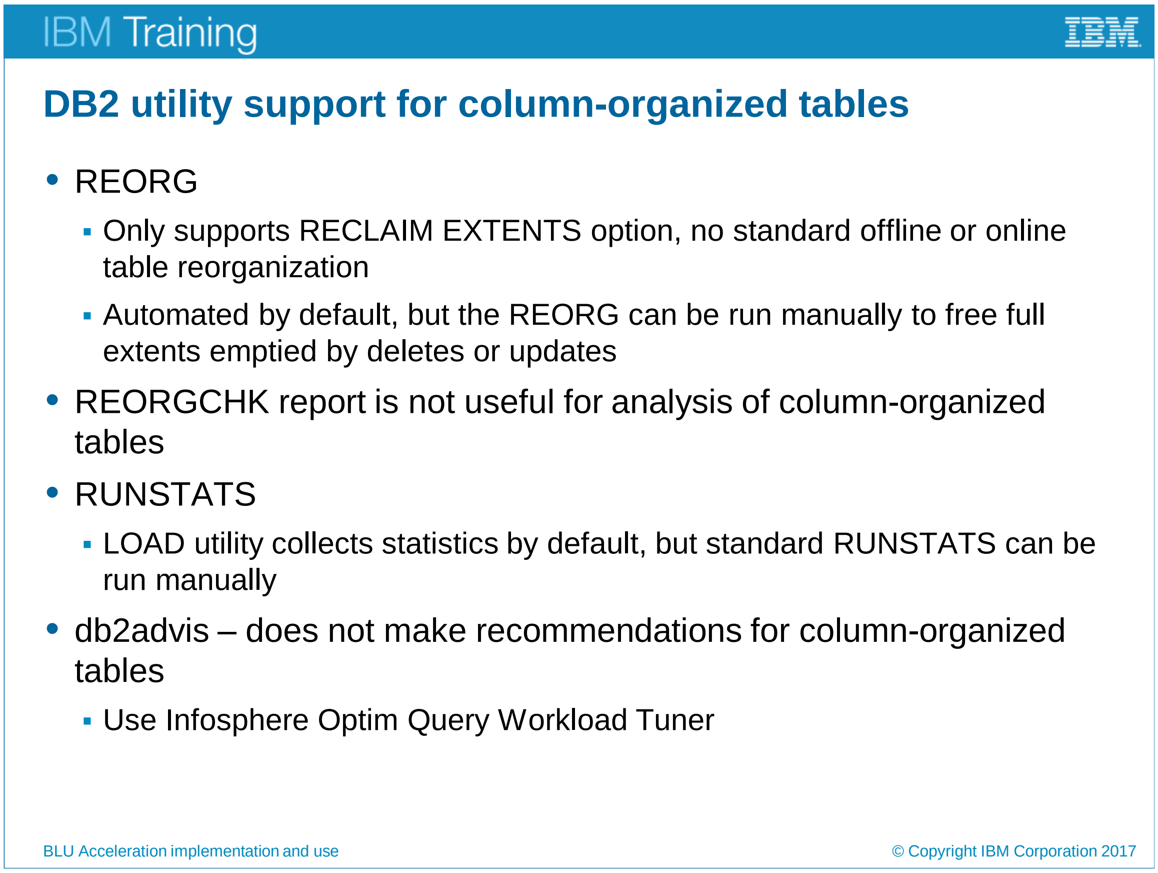
Use the ADMIN_MOVE_TABLE stored procedure to convert the row-organized STAFF table into a column-organized table without specifying a target table. The **ORGANIZE BY** COLUMN clause must be specified as a parameter so that the target table is created as a column-organized table.

```
CALL SYSPROC.ADMIN_MOVE_TABLE(
'OTM01COL',
'STAFF',
'',
'',
'',
'ORGANIZE BY COLUMN',
'',
'',
'',
'COPY_USE_LOAD',
'MOVE'
)
```

In the following example a table named TEST.ACCT2 is moved to a new tablespace and converted to a column-organized table. The LOAD utility is used for the COPY phase.

```
call admin_move_table('TEST','ACCT2','AS2','AS2','AS2'
,'ORGANIZE BY COLUMN', '','','','COPY_USE_LOAD','MOVE')
Result set 1
--------------
KEY                              VALUE
-------------------------------- --------------------------------------
-----------------------------------------------------------------------
-----------------
AUTHID                           INST20
CLEANUP_END                      2013-06-26-12.13.36.463983
CLEANUP_START                    2013-06-26-12.13.36.275748
COPY_END                         2013-06-26-12.13.35.640270
COPY_OPTS
OVER_INDEX,LOAD,WITH_INDEXES,NON_CLUSTER
COPY_START                       2013-06-26-12.13.21.951896
COPY_TOTAL_ROWS                  1000000
```

```
INDEXNAME                        ACCT2ACCT
INDEXSCHEMA                      TEST
INDEX_CREATION_TOTAL_TIME        0
INIT_END                         2013-06-26-12.13.21.537935
INIT_START                       2013-06-26-12.13.19.632628
ORIGINAL_TBLSIZE                 129280
REPLAY_END                       2013-06-26-12.13.36.163455
REPLAY_START                     2013-06-26-12.13.35.640947
REPLAY_TOTAL_ROWS                0
REPLAY_TOTAL_TIME                0
STATUS                           COMPLETE
SWAP_END                         2013-06-26-12.13.36.229232
SWAP_RETRIES                     0
SWAP_START                       2013-06-26-12.13.36.164119
UTILITY_INVOCATION_ID
0100000006000000080000000000000000201306261213215394390000000
VERSION                          10.05.0000
23 record(s) selected.
Return Status = 0
```


IBM Training

DB2 utility support for column-organized tables

- REORG
  - Only supports RECLAIM EXTENTS option, no standard offline or online table reorganization
  - Automated by default, but the REORG can be run manually to free full extents emptied by deletes or updates
- REORGCHK report is not useful for analysis of column-organized tables
- RUNSTATS
  - LOAD utility collects statistics by default, but standard RUNSTATS can be run manually
- db2advis – does not make recommendations for column-organized tables
  - Use Infosphere Optim Query Workload Tuner

BLU Acceleration implementation and use

© Copyright IBM Corporation 2017

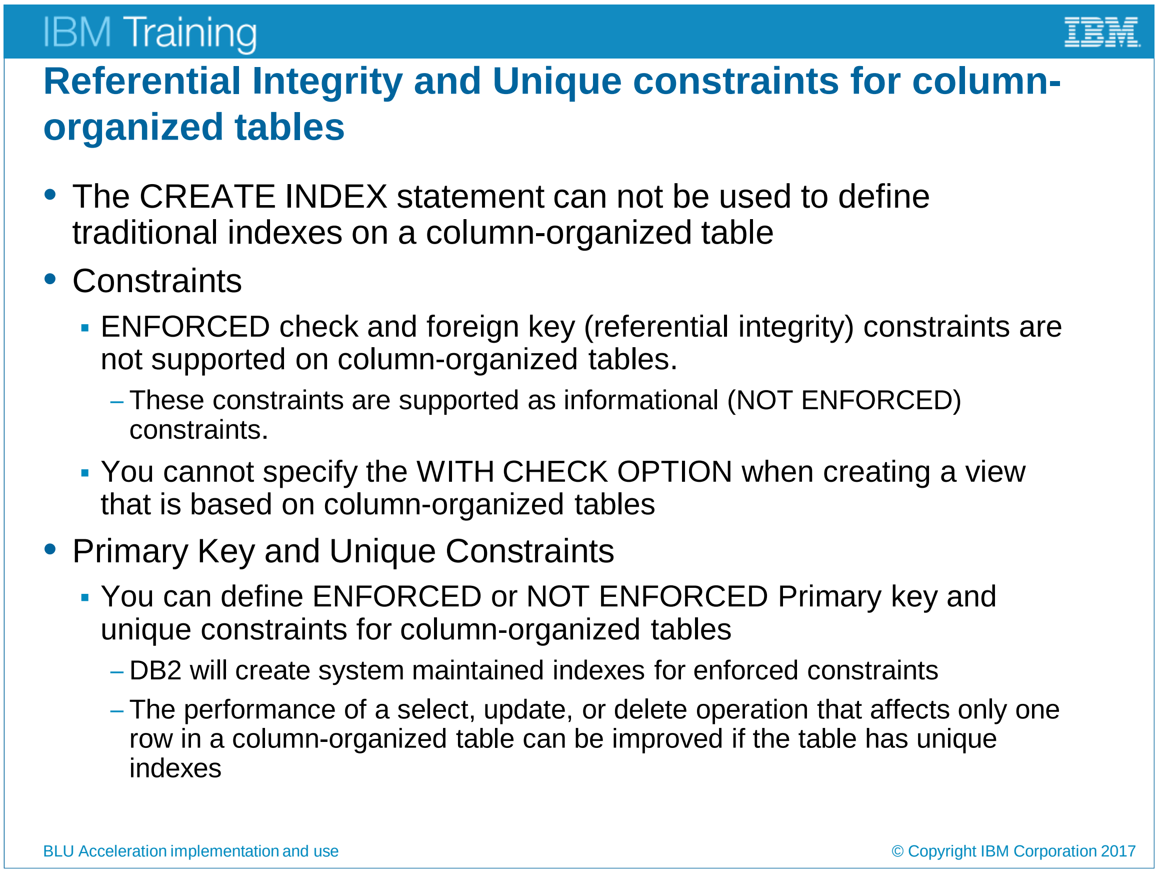
DB2 utility support for column-organized tables

With DB2 11 the only mode of processing for the REORG utility with column-organized tables is the RECLAIM EXTENTS option. When DB2_WORKLOAD is set to ANALYTICS, DB2 will automatically run the RECLAIM EXTENTS type of reorganization for column-organized tables.

The REORGCHK command report does not provide useful analysis for column-organized tables.

The LOAD utility will by default collect table statistics when an empty column-organized table is loaded. The RUNSTATS command can also be run manually to collect new table statistics.

When providing recommendations about indexes, MQTs, or MDC tables, the Design Advisor, db2advis, ignores column-organized tables.



IBM Training

Referential Integrity and Unique constraints for column-organized tables

- The CREATE INDEX statement can not be used to define traditional indexes on a column-organized table
- Constraints
  - ENFORCED check and foreign key (referential integrity) constraints are not supported on column-organized tables.
    - These constraints are supported as informational (NOT ENFORCED) constraints.
  - You cannot specify the WITH CHECK OPTION when creating a view that is based on column-organized tables
- Primary Key and Unique Constraints
  - You can define ENFORCED or NOT ENFORCED Primary key and unique constraints for column-organized tables
    - DB2 will create system maintained indexes for enforced constraints
    - The performance of a select, update, or delete operation that affects only one row in a column-organized table can be improved if the table has unique indexes

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Referential Integrity and Unique constraints for column-organized tables

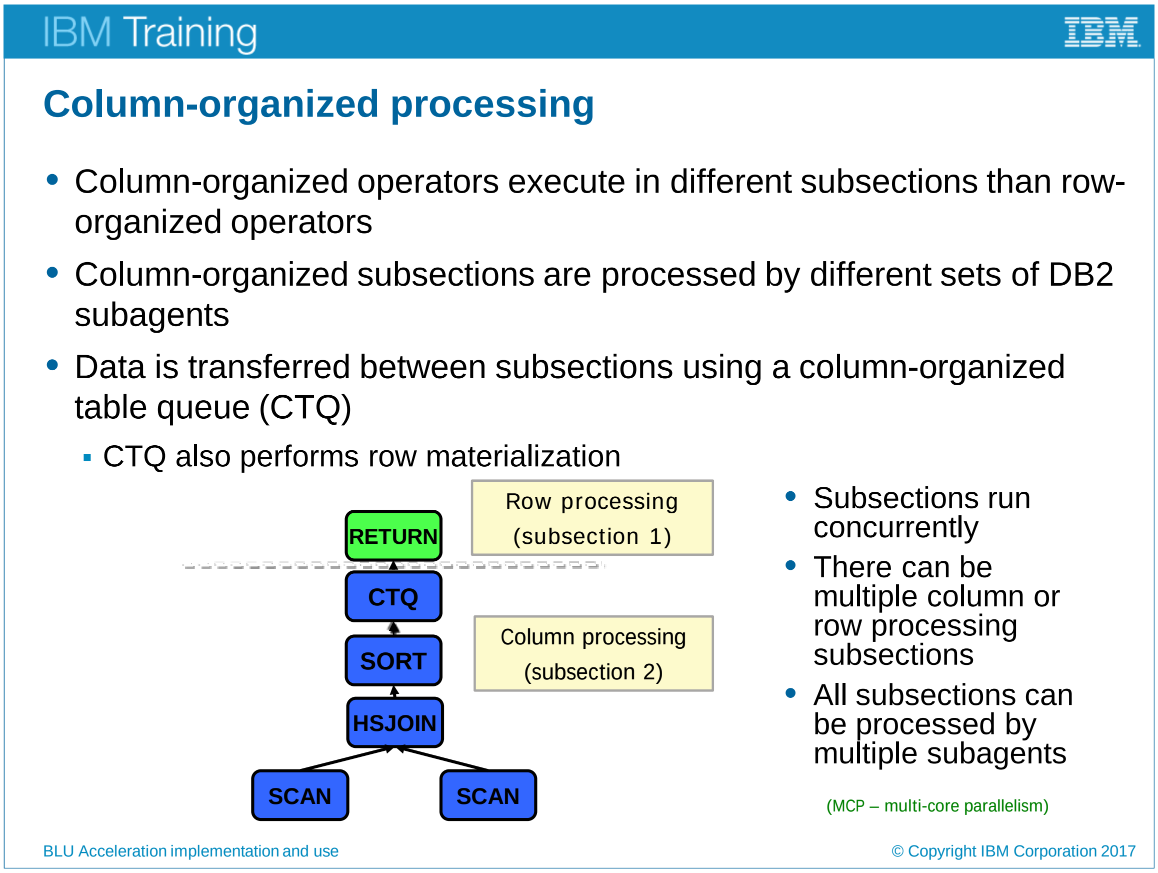
The CREATE INDEX statement is not supported to define an index for a column-organized table.

ENFORCED check and foreign key (referential integrity) constraints are not supported on column-organized tables. These constraints are supported as informational (NOT ENFORCED) constraints.

You cannot specify the WITH CHECK OPTION when creating a view that is based on column-organized tables.

You can define ENFORCED and NOT ENFORCED Primary Key and Unique constraints for column organized tables. If you define an enforced Primary Key or Unique constraint for a column organized table, DB2 will create the necessary supporting index that will be used to perform the necessary checking when rows are added or updated.

The performance of a select, update, or delete operation that affects only one row in a column-organized table can be improved if the table has unique indexes because the query optimizer can use an index scan instead of a full table scan.



IBM Training

Column-organized processing

- Column-organized operators execute in different subsections than row-organized operators
- Column-organized subsections are processed by different sets of DB2 subagents
- Data is transferred between subsections using a column-organized table queue (CTQ)
  - CTQ also performs row materialization

RETURN
Row processing (subsection 1)
CTQ
SORT
Column processing (subsection 2)
HSJOIN
SCAN
SCAN

- Subsections run concurrently
- There can be multiple column or row processing subsections
- All subsections can be processed by multiple subagents

(MCP – multi-core parallelism)

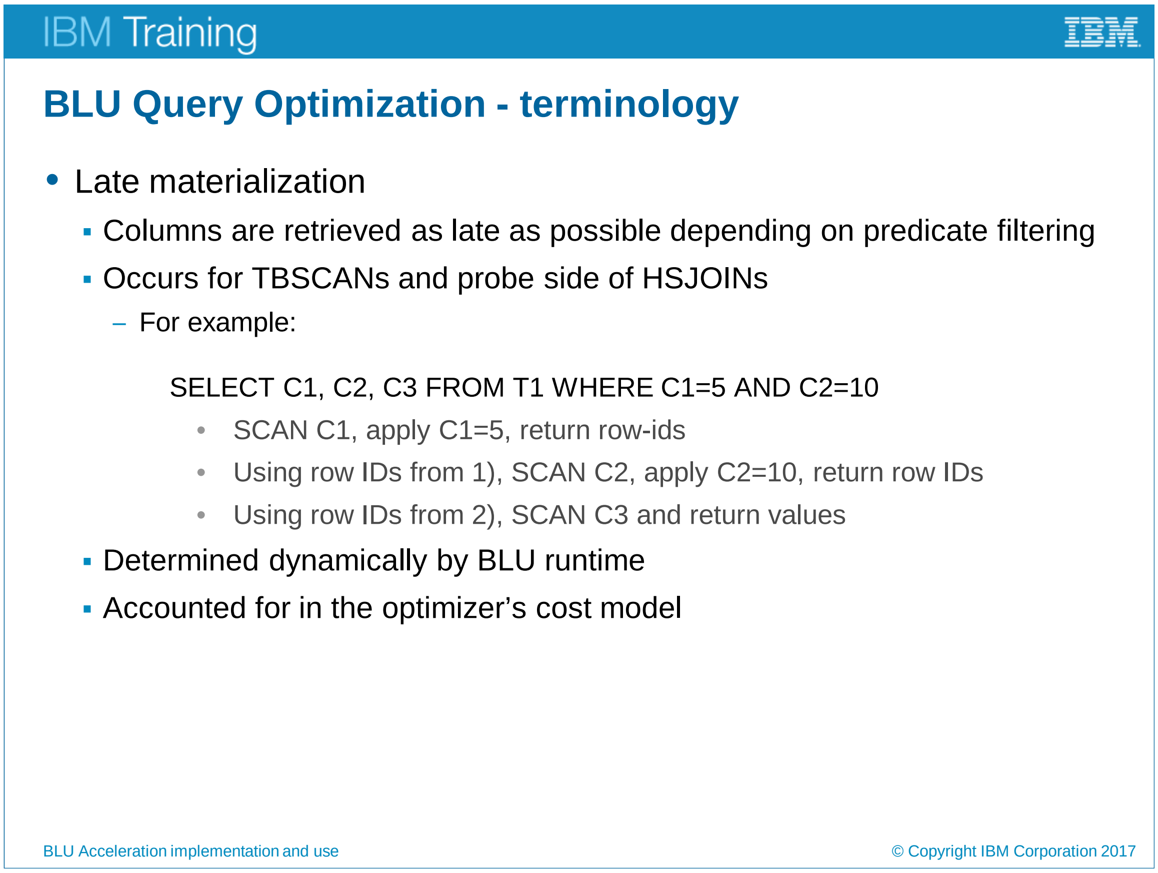
BLU Acceleration implementation and use

© Copyright IBM Corporation 2017

Column-organized processing

The CTQ operator represents a boundary within the DB2 query engine. Operators that appear below the boundary process data as compressed column-organized vectors and tuples, whereas operators that are above the boundary operate on tuples that are not encoded.

With DB2 10.5, some of the common processing functions were implemented into the column processing, such as the hash join operation. DB2 11.1 provides many new operations, including sorts and nested-loop join processing that can be part of the column-organized processing subsections.


IBM Training

BLU Query Optimization - terminology

- Late materialization
  - Columns are retrieved as late as possible depending on predicate filtering
  - Occurs for TBSCANs and probe side of HSJOINs
    - For example:

      SELECT C1, C2, C3 FROM T1 WHERE C1=5 AND C2=10
      - SCAN C1, apply C1=5, return row-ids
      - Using row IDs from 1), SCAN C2, apply C2=10, return row IDs
      - Using row IDs from 2), SCAN C3 and return values
  - Determined dynamically by BLU runtime
  - Accounted for in the optimizer’s cost model

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

BLU Query Optimization - terminology

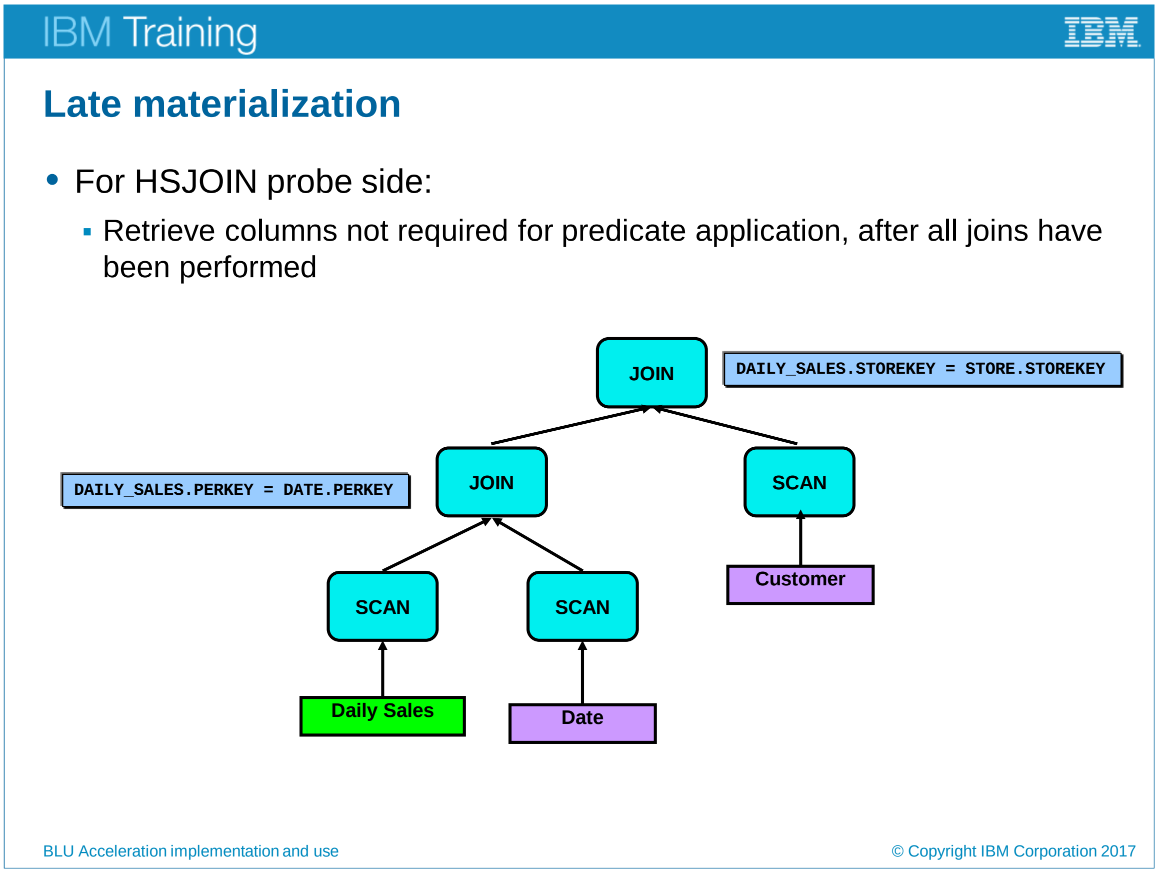
The term late materialization is the concept implemented for BLU Acceleration in DB2 10.5 and extended with DB2 11.1, that DB2 will perform as much of the SQL processing as possible in the access plan using the highly compressed, encoded form of the column data. Column data that is determined necessary to produce the result will be unencoded late in the access plan to reduce processing costs.

Consider, for example, this simple SQL statement:

```
SELECT C1, C2, C3 FROM T1 WHERE C1=5 AND C2 = 10
```

Rather than scan the table T1, decoding large numbers of data rows to apply the predicates, with column organized processing, DB2 could:

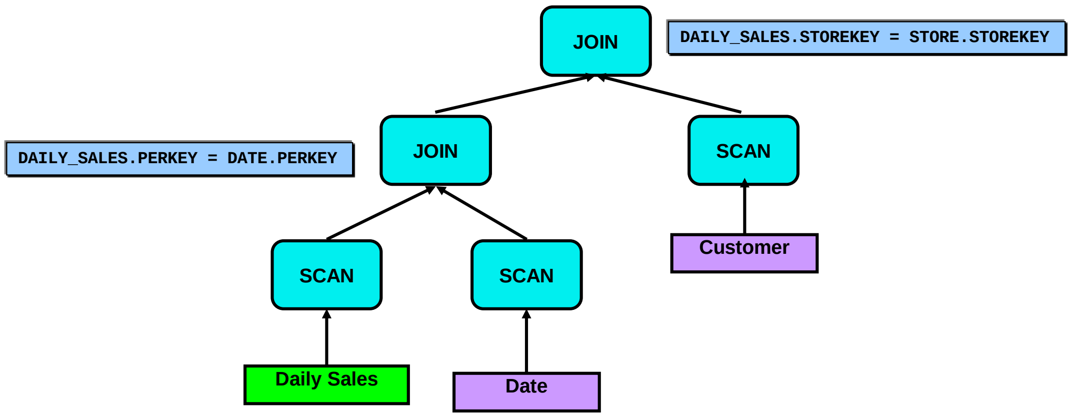
1. Scan the encoded data for column C1 using the predicate C1 = 5, and produce a set of row ids.
2. Utilize the set of row ids from step 1 to scan column C2 using the predicate C2 = 10, and produce a smaller set of row ids.
3. Utilize the reduced set of row ids from step 2 to scan column C3. The encoded values for C3 will be unencoded to produce the final result.



IBM Training

Late materialization

- For HSJOIN probe side:
  - Retrieve columns not required for predicate application, after all joins have been performed


JOIN
DAILY_SALES.STOREKEY = STORE.STOREKEY
JOIN
SCAN
DAILY_SALES.PERKEY = DATE.PERKEY
SCAN
SCAN
Customer
Daily Sales
Date

BLU Acceleration implementation and use

© Copyright IBM Corporation 2017

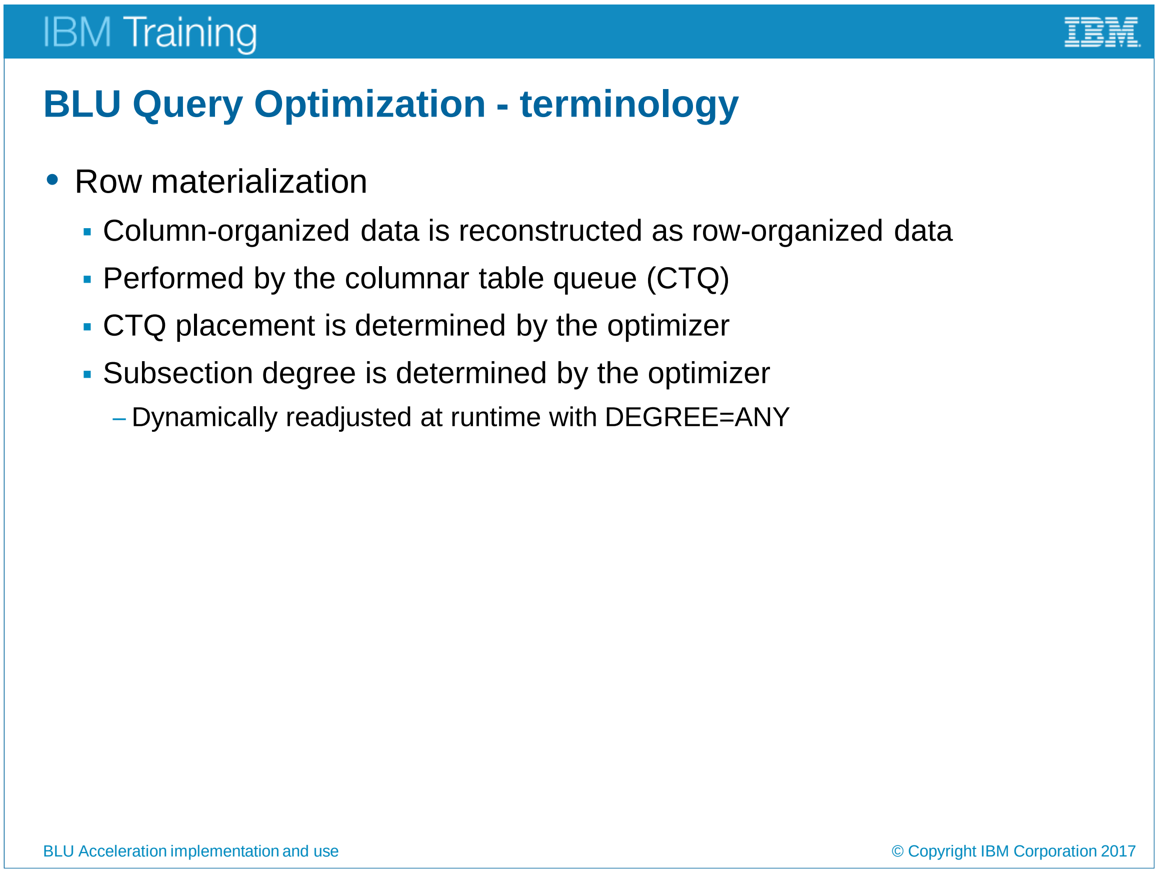
Late materialization

The concept of late materialization can be used when multiple tables are joined.

The join operations may require one or more columns from the tables to be utilized to evaluate the join predicates.

In the slide example, three tables are being joined. The access plan diagram shows the Date and Daily Sales tables being joined with an equality predicate using the PERKEY column from each table. The composite data from that join are joined to the Customer table using an equality predicate based on the STOREKEY column data from the Daily Sales and Customer tables.

The columns from the three tables that were not needed to perform the join processing are retrieved and decoded for the subset of rows represented in the join results.



IBM Training

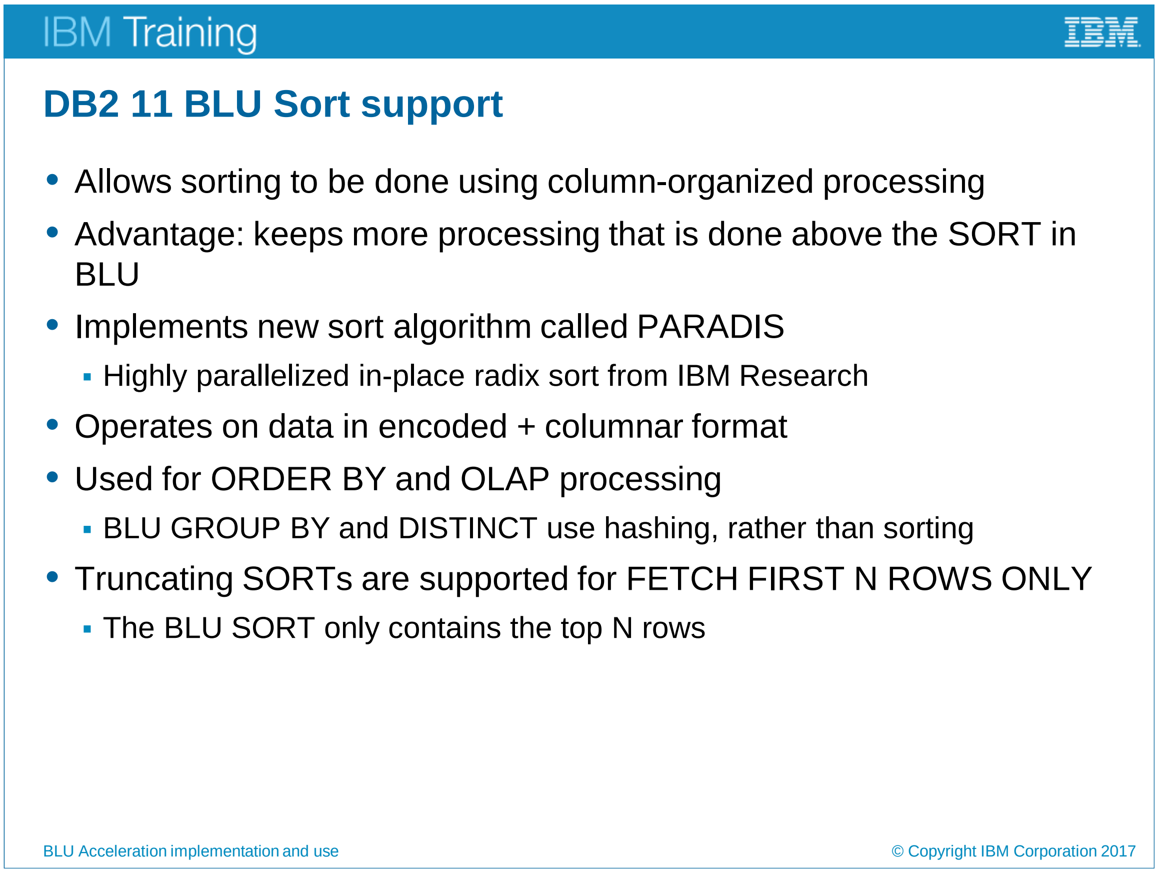
BLU Query Optimization - terminology

- Row materialization
  - Column-organized data is reconstructed as row-organized data
  - Performed by the columnar table queue (CTQ)
  - CTQ placement is determined by the optimizer
  - Subsection degree is determined by the optimizer
    - Dynamically readjusted at runtime with DEGREE=ANY

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

BLU Query Optimization - terminology

The materialization of result rows is part of the processing of the CTQ operators in the access plan. These are placed in the access plan by the DB2 optimizer. Other access plan options, such as the degree of parallelism, are also selected by the optimizer.



IBM Training

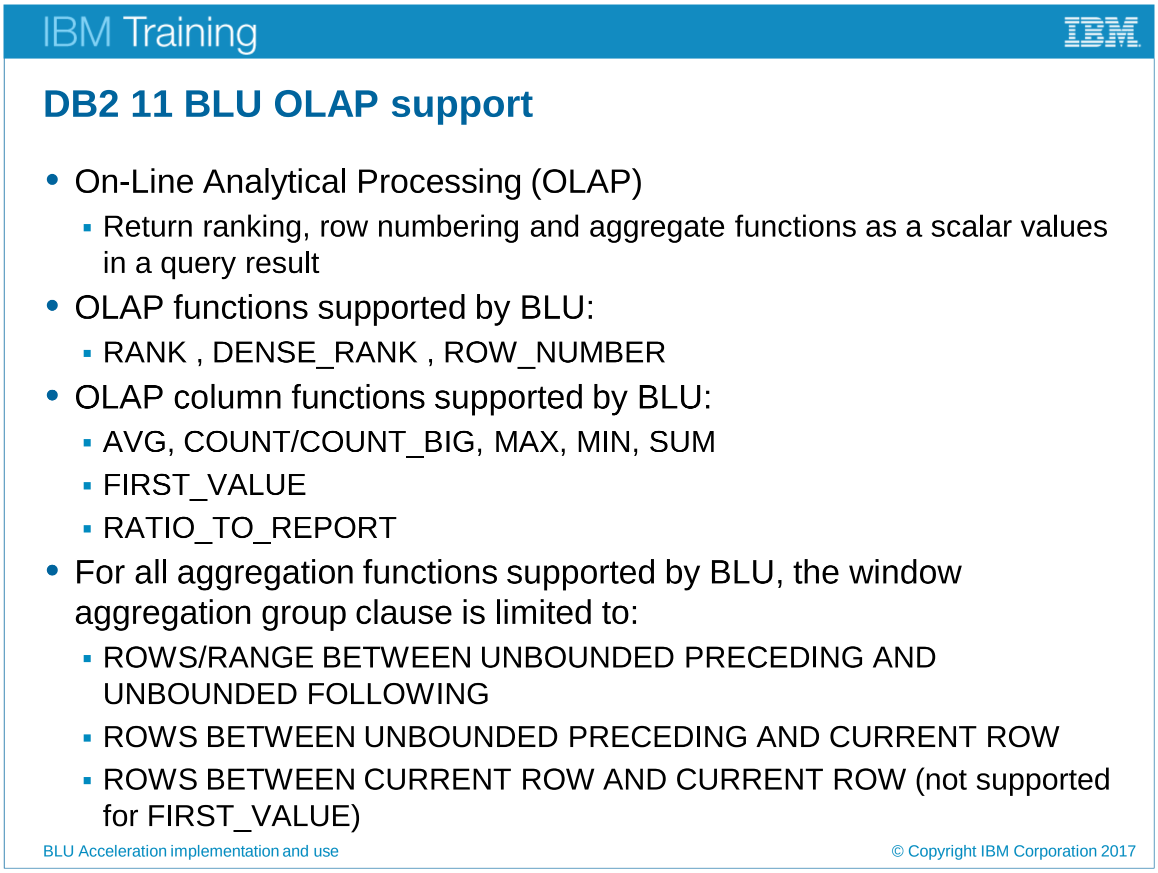
DB2 11 BLU Sort support

- Allows sorting to be done using column-organized processing
- Advantage: keeps more processing that is done above the SORT in BLU
- Implements new sort algorithm called PARADIS
  - Highly parallelized in-place radix sort from IBM Research
- Operates on data in encoded + columnar format
- Used for ORDER BY and OLAP processing
  - BLU GROUP BY and DISTINCT use hashing, rather than sorting
- Truncating SORTs are supported for FETCH FIRST N ROWS ONLY
  - The BLU SORT only contains the top N rows

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

DB2 11 BLU Sort support

With DB2 11, the BLU Sort function utilized the industry-leading parallel sort algorithm, which leverages the latest innovations from the IBM Thomas J. Watson Research Center. This parallel sort features a fast radix sort with superior parallelism that is able to sort compressed and encoded data.


IBM Training

DB2 11 BLU OLAP support

- On-Line Analytical Processing (OLAP)
  - Return ranking, row numbering and aggregate functions as a scalar values in a query result
- OLAP functions supported by BLU:
  - RANK , DENSE_RANK , ROW_NUMBER
- OLAP column functions supported by BLU:
  - AVG, COUNT/COUNT_BIG, MAX, MIN, SUM
  - FIRST_VALUE
  - RATIO_TO_REPORT
- For all aggregation functions supported by BLU, the window aggregation group clause is limited to:
  - ROWS/RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
  - ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
  - ROWS BETWEEN CURRENT ROW AND CURRENT ROW (not supported for FIRST_VALUE)

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

DB2 11 BLU OLAP support

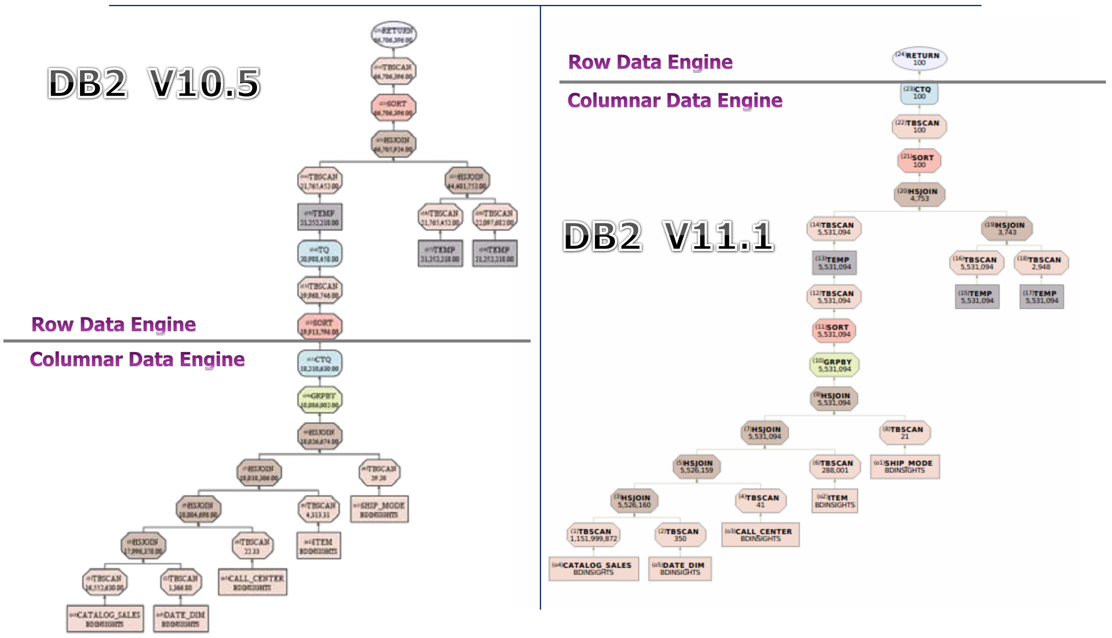
On-Line Analytical Processing (OLAP) functions provide the ability to return ranking, row numbering, and existing aggregate function information as a scalar value in a query result.

The slide lists a number of common OLAP functions that are now supported by the column organized processing portion of SQL processing with DB2 11.

There are some limitations on aggregation functions supported by the BLU processing engine, as indicated on the slide.

IBM Training

Native sort + OLAP support - Access plan difference with Native Evaluator support

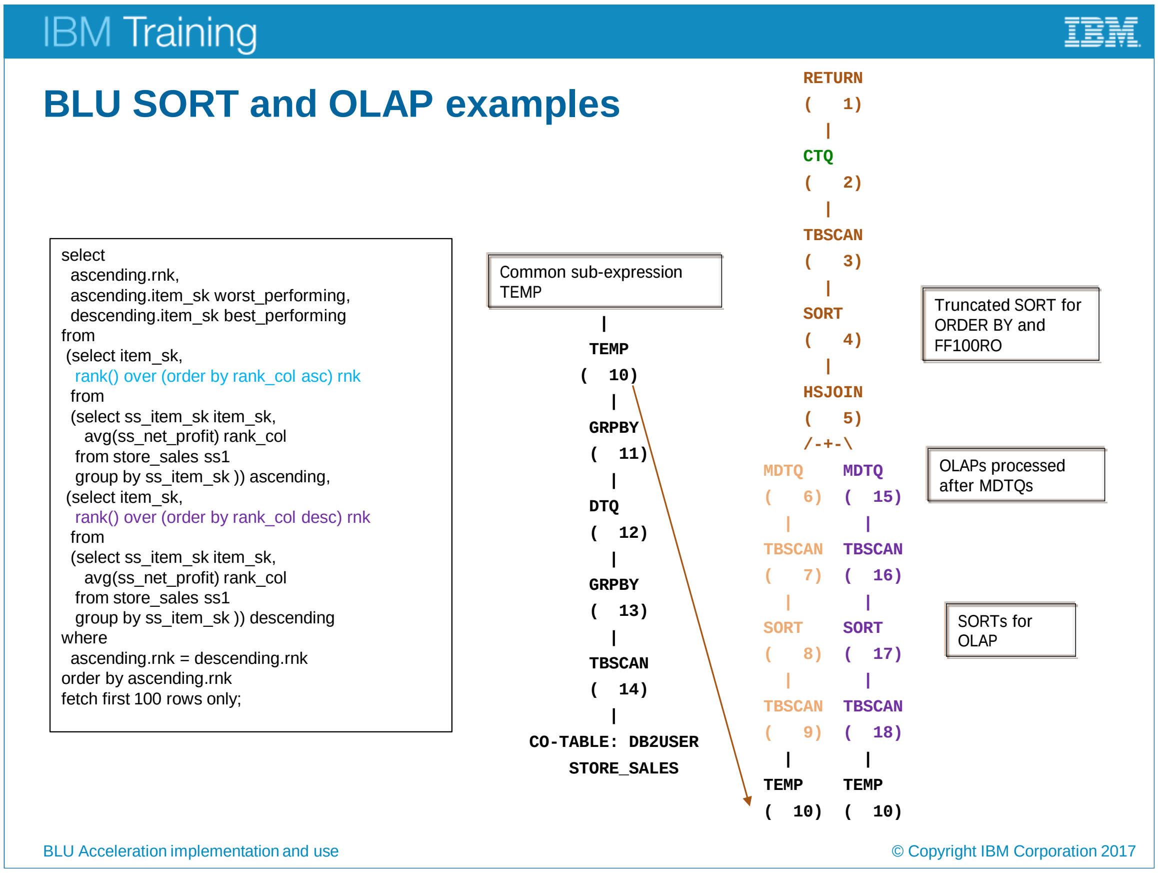


BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Native sort + OLAP support - Access plan difference with Native Evaluator support

The enhancements for native sort and OLAP function processing in the column-organized processing section of the access plans can be seen using the access plan reports from DB2 explain tools.

The access plans will show what part of the processing is done in the row engine versus the BLU engine. All parts below the "CTQ" evaluator are done in the BLU engine. In the slide, we can see that by pushing down the SORT evaluator we keep the data processing in BLU (everything except the RETURN operation). For this sample SQL query, a 4x speedup was observed.



IBM Training

BLU SORT and OLAP examples

```
select
  ascending.rnk,
  ascending.item_sk worst_performing,
  descending.item_sk best_performing
from
 (select item_sk,
   rank() over (order by rank_col asc) rnk
  from
  (select ss_item_sk item_sk,
     avg(ss_net_profit) rank_col
   from store_sales ss1
   group by ss_item_sk )) ascending,
 (select item_sk,
   rank() over (order by rank_col desc) rnk
  from
  (select ss_item_sk item_sk,
     avg(ss_net_profit) rank_col
   from store_sales ss1
   group by ss_item_sk )) descending
where
  ascending.rnk = descending.rnk
order by ascending.rnk
fetch first 100 rows only;
```

Common sub-expression TEMP

```
            |
          TEMP
         (  10)
            |
          GRPBY
         (  11)
            |
           DTQ
         (  12)
            |
          GRPBY
         (  13)
            |
          TBSCAN
         (  14)
            |
   CO-TABLE: DB2USER
       STORE_SALES
```

```
          RETURN
          (   1)
            |
          CTQ
          (   2)
            |
          TBSCAN
          (   3)
            |
          SORT
          (   4)
            |
          HSJOIN
          (   5)
          /-+-\
    MDTQ       MDTQ
    (   6)     (  15)
      |          |
    TBSCAN     TBSCAN
    (   7)     (  16)
      |          |
    SORT       SORT
    (   8)     (  17)
      |          |
    TBSCAN     TBSCAN
    (   9)     (  18)
      |          |
    TEMP       TEMP
    (  10)     (  10)
```

Truncated SORT for ORDER BY and FF100RO

OLAPs processed after MDTQs

SORTs for OLAP

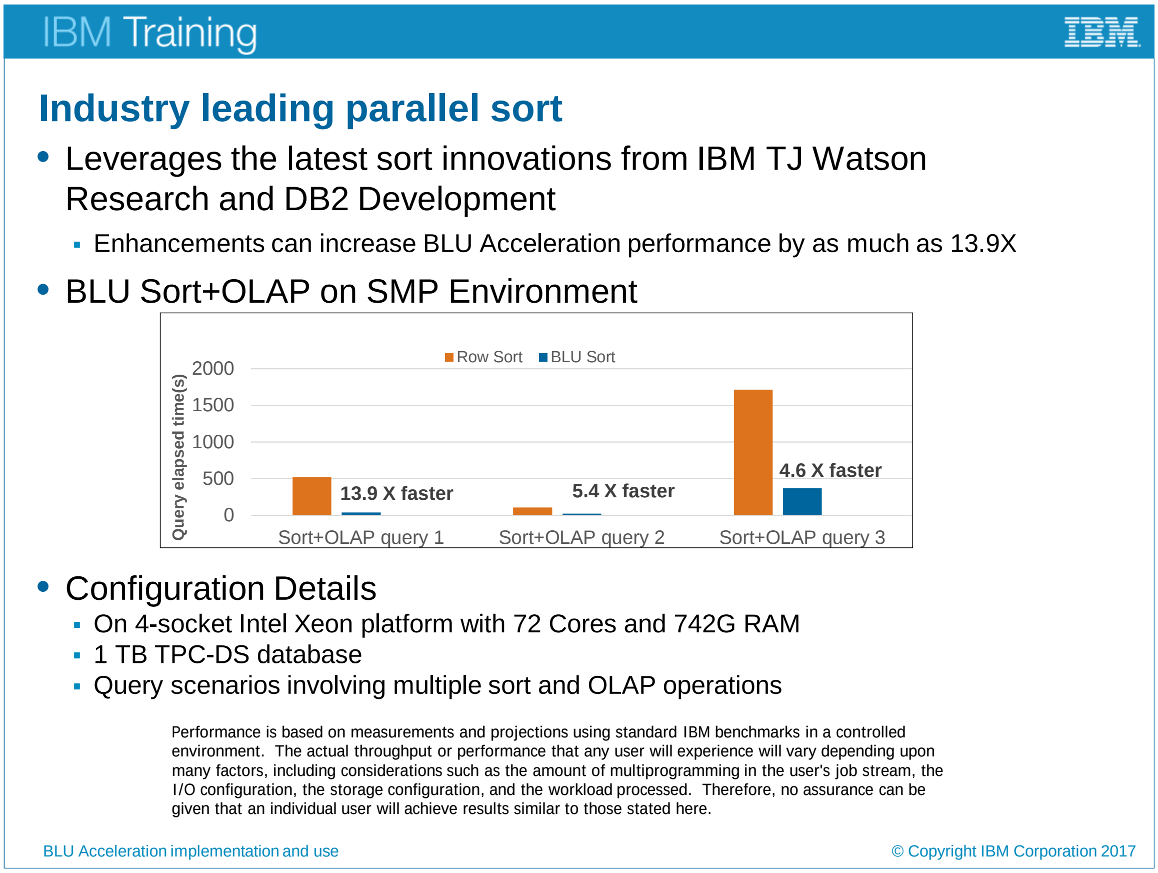
BLU Acceleration implementation and use © Copyright IBM Corporation 2017

BLU SORT and OLAP examples

The example query on this page is computing the top best and worst performing items based on average net profit.

The RANK OLAP function is used to assign a rank based on the ascending and descending order of average net profit. The two ranks are then joined, the final result is sorted on rank, and the top/bottom 100 items are returned.

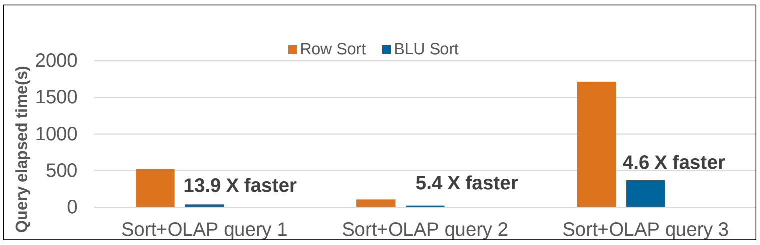
The explain output shows how all operations are performed using column-organized processing because there is only one CTQ operation at the top of the plan. The example also shows a common sub-expression temp that allows the result of the average to be shared for the computation of each of the RANK functions.



IBM Training

Industry leading parallel sort

- Leverages the latest sort innovations from IBM TJ Watson Research and DB2 Development
  - Enhancements can increase BLU Acceleration performance by as much as 13.9X
- BLU Sort+OLAP on SMP Environment



- Configuration Details
  - On 4-socket Intel Xeon platform with 72 Cores and 742G RAM
  - 1 TB TPC-DS database
  - Query scenarios involving multiple sort and OLAP operations

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.

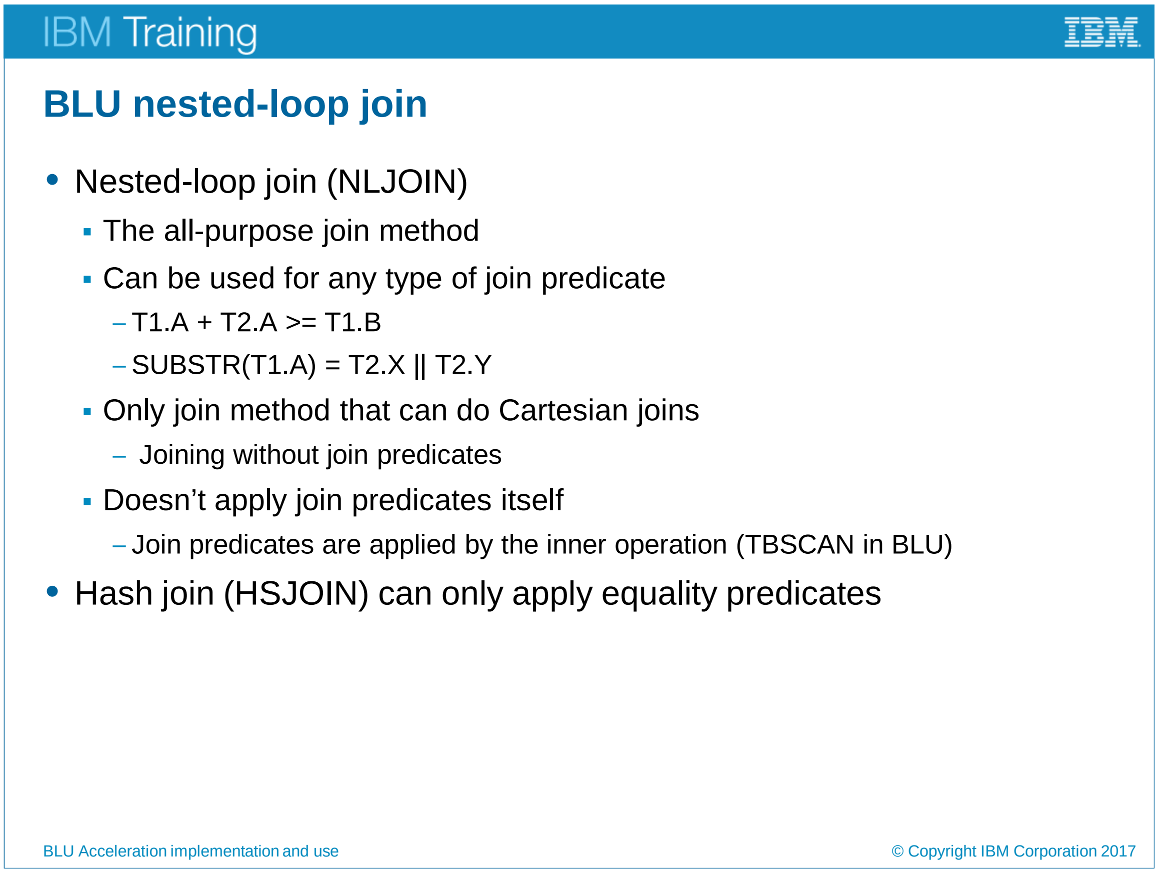
BLU Acceleration implementation and use

© Copyright IBM Corporation 2017

Industry leading parallel sort

The slide shows the impact on query elapsed times for three SQL queries that include multiple sort and OLAP functions.

The first bar in each group shows the performance using the row-processing sort that was utilized with DB2 10.5, while the second bar shows the improved performance using the column-organized sort and OLAP processing in DB2 11.1.



IBM Training

BLU nested-loop join

- Nested-loop join (NLJOIN)
  - The all-purpose join method
  - Can be used for any type of join predicate
    - T1.A + T2.A >= T1.B
    - SUBSTR(T1.A) = T2.X || T2.Y
  - Only join method that can do Cartesian joins
    - Joining without join predicates
  - Doesn’t apply join predicates itself
    - Join predicates are applied by the inner operation (TBSCAN in BLU)
- Hash join (HSJOIN) can only apply equality predicates

BLU Acceleration implementation and use

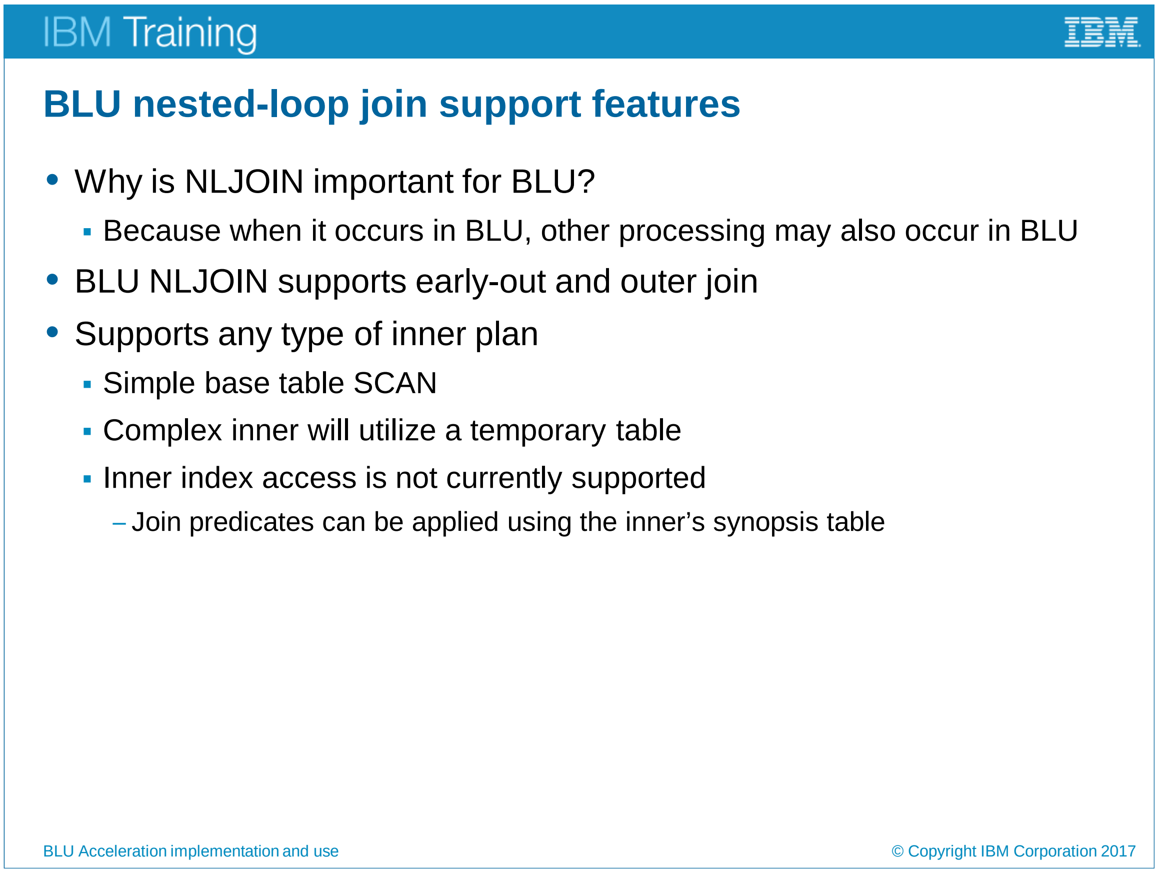
© Copyright IBM Corporation 2017

BLU nested-loop join

With DB2 10.5, the column-organized processing engine was only able to perform table joins using the Hash Join method, which requires an equality predicate between the tables being joined.

The slide lists some types of join predicates that can utilize a nested-loop join, including Cartesian joins.

With DB2 11.1, a nested-loop join operation can also be performed by the CDE engine.



IBM Training

BLU nested-loop join support features

- Why is NLJOIN important for BLU?
  - Because when it occurs in BLU, other processing may also occur in BLU
- BLU NLJOIN supports early-out and outer join
- Supports any type of inner plan
  - Simple base table SCAN
  - Complex inner will utilize a temporary table
  - Inner index access is not currently supported
    - Join predicates can be applied using the inner’s synopsis table

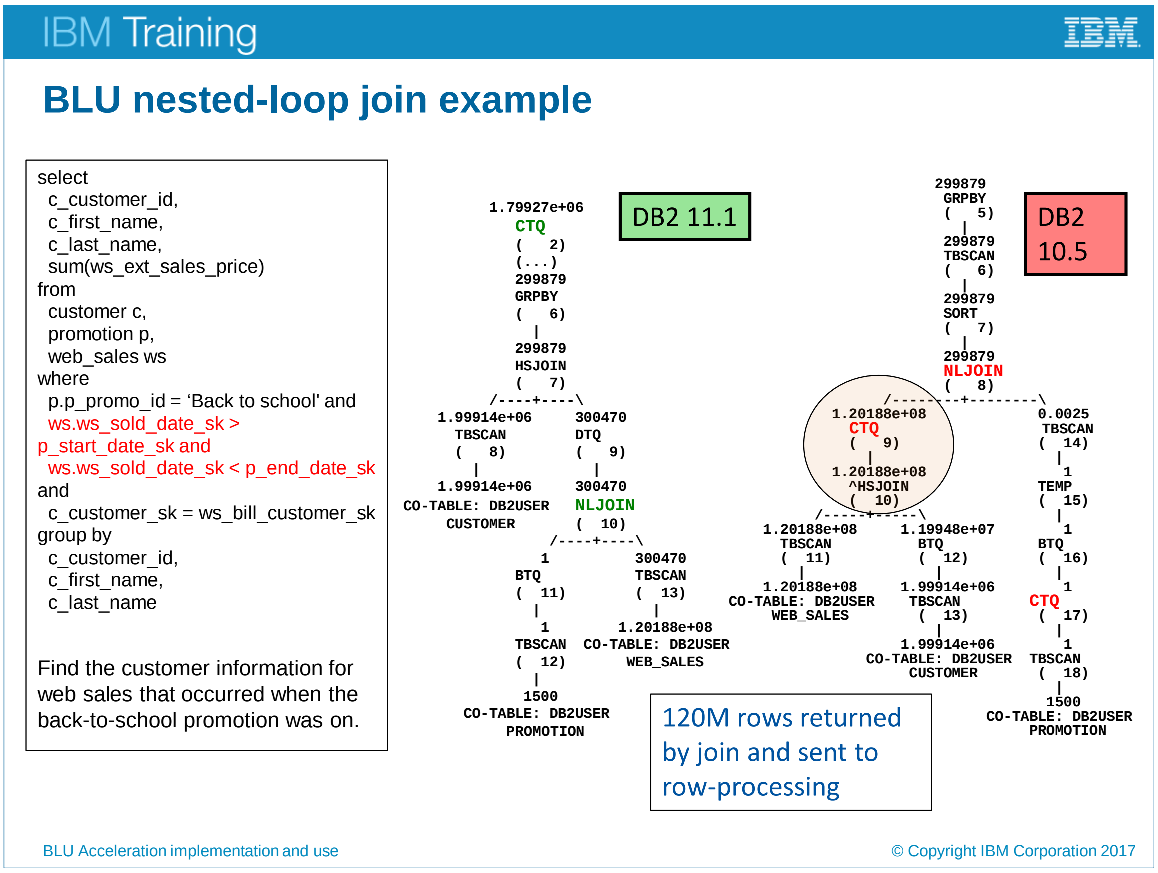
BLU Acceleration implementation and use

© Copyright IBM Corporation 2017

BLU nested-loop join support features

The scan performed for the inner table of the nested-loop join will utilize a table scan, but it could make use of a temporary table for some complex inners.

In DB2 11.1, index based scans are not supported for NLJ inner tables.



IBM Training

BLU nested-loop join example

```
select
  c_customer_id,
  c_first_name,
  c_last_name,
  sum(ws_ext_sales_price)
from
  customer c,
  promotion p,
  web_sales ws
where
  p.p_promo_id = 'Back to school' and
  ws.ws_sold_date_sk >
p_start_date_sk and
  ws.ws_sold_date_sk < p_end_date_sk
and
  c_customer_sk = ws_bill_customer_sk
group by
  c_customer_id,
  c_first_name,
  c_last_name
```

Find the customer information for web sales that occurred when the back-to-school promotion was on.

DB2 11.1

```
              1.79927e+06
                 CTQ
                 (   2)
                 (...)
                 299879
                 GRPBY
                 (   6)
                   |
                 299879
                 HSJOIN
                 (   7)
              /----+----\
      1.99914e+06       300470
        TBSCAN          DTQ
        (   8)          (   9)
          |               |
      1.99914e+06       300470
  CO-TABLE: DB2USER     NLJOIN
       CUSTOMER         (  10)
                     /----+----\
                    1          300470
                 BTQ           TBSCAN
                 (  11)        (  13)
                   |             |
                    1        1.20188e+08
                 TBSCAN    CO-TABLE: DB2USER
                 (  12)       WEB_SALES
                   |
                  1500
           CO-TABLE: DB2USER
              PROMOTION
```

DB2 10.5

```
                         299879
                          GRPBY
                          (   5)
                           |
                          299879
                          TBSCAN
                          (   6)
                           |
                          299879
                          SORT
                          (   7)
                           |
                          299879
                          NLJOIN
                          (   8)
                /--------+--------\
        1.20188e+08                0.0025
            CTQ                    TBSCAN
            (   9)                 (  14)
             |                       |
        1.20188e+08                  1
          ^HSJOIN                  TEMP
          (  10)                   (  15)
       /-----+-----\                 |
 1.20188e+08     1.19948e+07         1
   TBSCAN          BTQ             BTQ
   (  11)          (  12)          (  16)
    |               |                |
 1.20188e+08     1.99914e+06         1
CO-TABLE: DB2USER  TBSCAN         CTQ
  WEB_SALES        (  13)         (  17)
                    |                |
                 1.99914e+06         1
             CO-TABLE: DB2USER   TBSCAN
                  CUSTOMER        (  18)
                                     |
                                    1500
                              CO-TABLE: DB2USER
                                 PROMOTION
```

120M rows returned by join and sent to row-processing

BLU Acceleration implementation and use

© Copyright IBM Corporation 2017

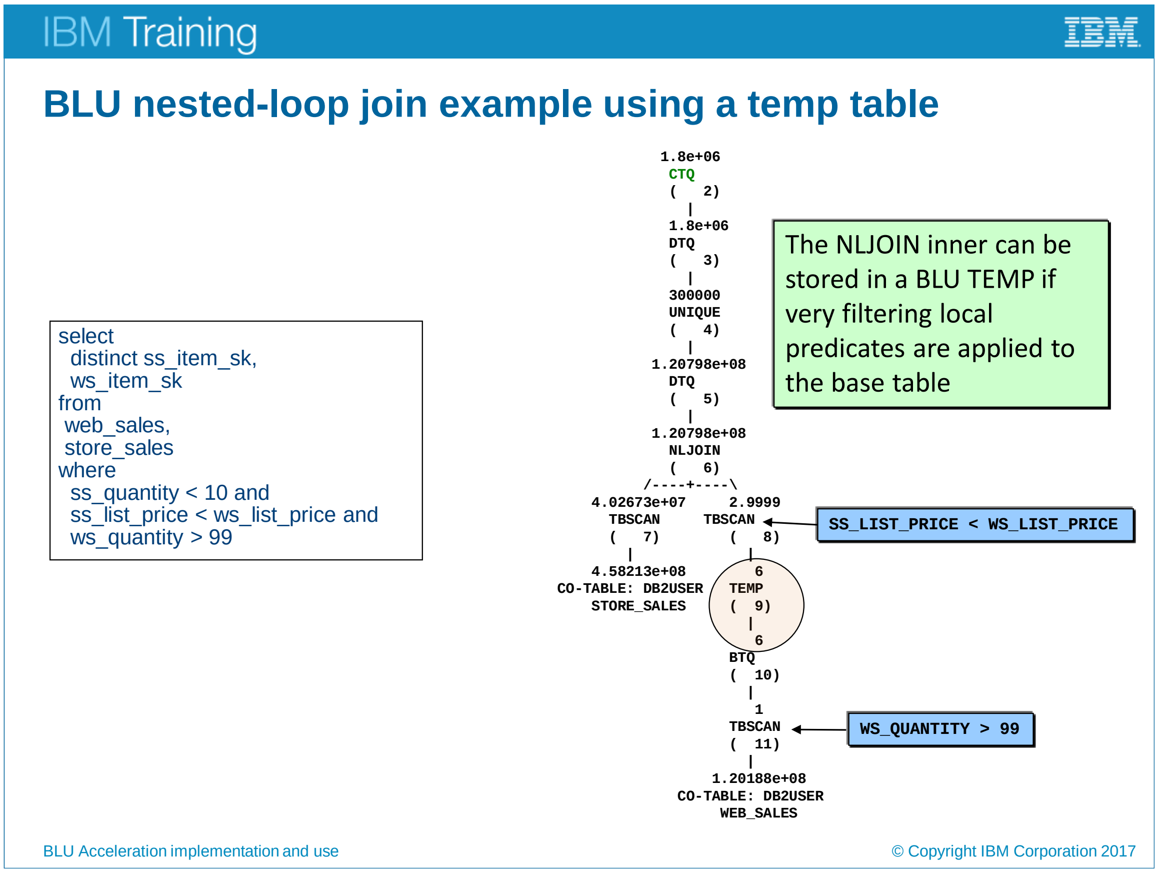
BLU nested-loop join example

This slide shows the DB2 10.5 and 11.1 access plans for a query that returns the customer information for Web sales that occurred when the back-to-school promotion was on.

The number in the access plans above the operator name represents the estimate of the number of rows returned by the operator.

In the 10.5 access plan, on the left, the result of the join between the WEB_SALES and the CUSTOMER tables is returned to row-organized processing using a column-organized table queue (CTQ). This join does not filter the WEB_SALES table so all the rows are returned, although they only include the WS_SOLD_DATE_SK and WS_EXT_SALES_PRICE columns. The single qualifying row from the PROMOTION table is also returned to row-organized processing where a nested-loop join is used to join it to the WEB_SALES data. Then the result is sorted in order to do the GROUP BY.

In DB2 11.1, the join between PROMOTION and WEB_SALES is performed in column-organized processing using a nested-loop join. This join filters the WEB_SALES table very well (120M -> 300K), reducing the cost of the join to the CUSTOMER table. The GROUP BY is also done in column-organized processing using a hash approach that avoids the need to sort.



IBM Training

BLU nested-loop join example using a temp table

```
select
  distinct ss_item_sk,
  ws_item_sk
from
 web_sales,
 store_sales
where
  ss_quantity < 10 and
  ss_list_price < ws_list_price and
  ws_quantity > 99
```

```
                  1.8e+06
                   CTQ
                   (   2)
                     |
                   1.8e+06
                   DTQ
                   (   3)
                     |
                   300000
                   UNIQUE
                   (   4)
                     |
                 1.20798e+08
                   DTQ
                   (   5)
                     |
                 1.20798e+08
                   NLJOIN
                   (   6)
                /----+----\
         4.02673e+07      2.9999
           TBSCAN         TBSCAN  <-- SS_LIST_PRICE < WS_LIST_PRICE
           (   7)         (   8)
             |              |
         4.58213e+08          6
     CO-TABLE: DB2USER    TEMP
         STORE_SALES      (   9)
                            |
                              6
                          BTQ
                          (  10)
                            |
                              1
                          TBSCAN  <-- WS_QUANTITY > 99
                          (  11)
                            |
                      1.20188e+08
                   CO-TABLE: DB2USER
                         WEB_SALES
```

The NLJOIN inner can be stored in a BLU TEMP if very filtering local predicates are applied to the base table

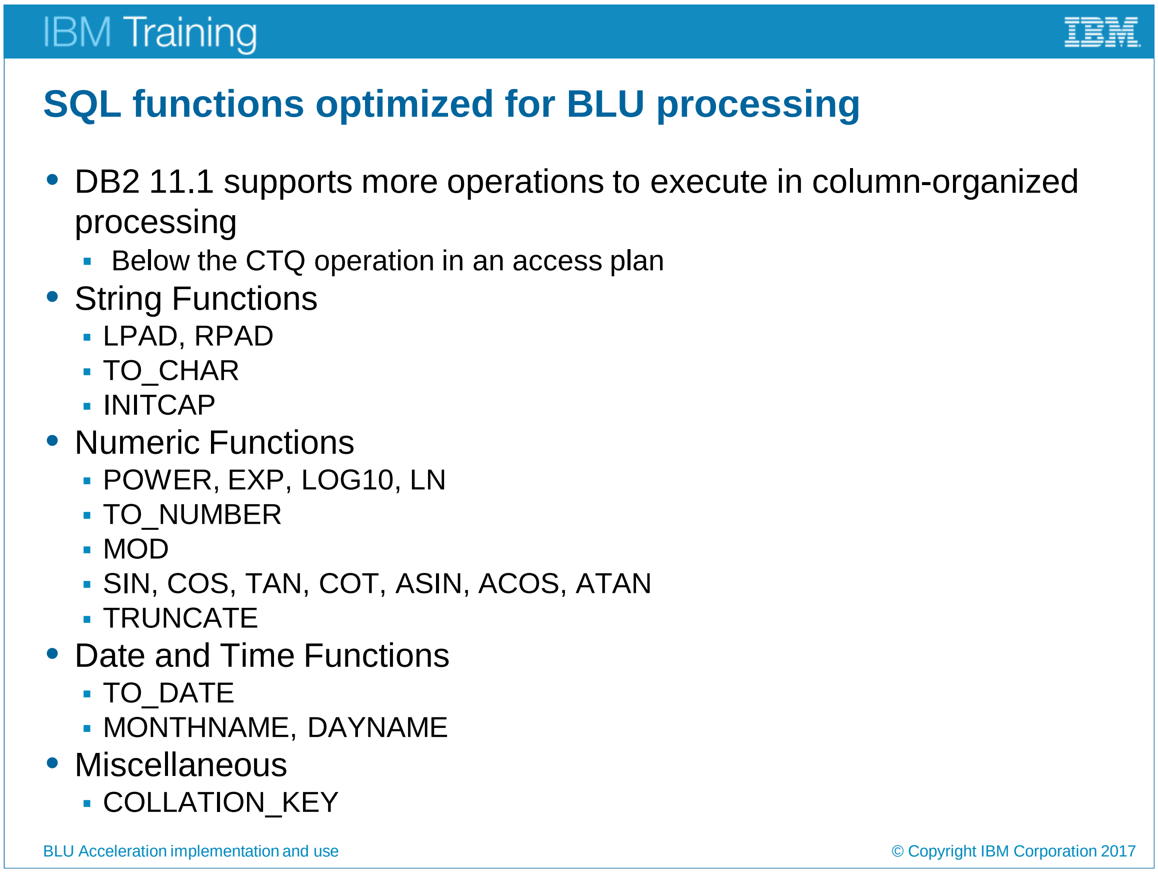
BLU Acceleration implementation and use

© Copyright IBM Corporation 2017

BLU nested-loop join example using a temp table

The slide shows an example of the access plan for the SQL query in the box on the left, produced using DB2 11.1.

The access plan includes a nested-loop join operation, performed by the column-organized engine. A BLU temporary table is utilized for the inner portion of the join to reduce costs. This temp would have the result of the table scan that applied the predicate "ws_quantity > 99".



IBM Training

SQL functions optimized for BLU processing

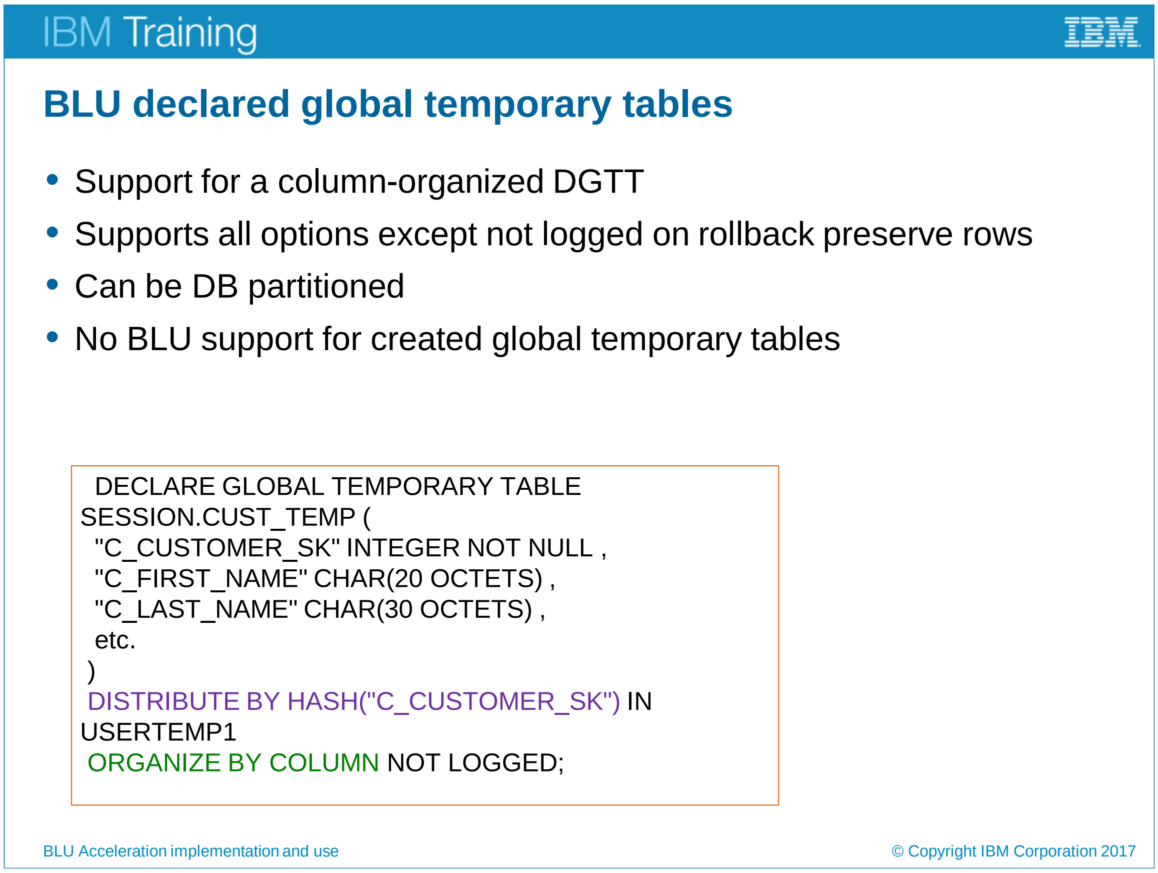
- DB2 11.1 supports more operations to execute in column-organized processing
  - Below the CTQ operation in an access plan
- String Functions
  - LPAD, RPAD
  - TO_CHAR
  - INITCAP
- Numeric Functions
  - POWER, EXP, LOG10, LN
  - TO_NUMBER
  - MOD
  - SIN, COS, TAN, COT, ASIN, ACOS, ATAN
  - TRUNCATE
- Date and Time Functions
  - TO_DATE
  - MONTHNAME, DAYNAME
- Miscellaneous
  - COLLATION_KEY

BLU Acceleration implementation and use

© Copyright IBM Corporation 2017

SQL functions optimized for BLU processing

The slide lists some of the string functions, numeric functions, and date and time functions that can be handled in the BLU engine with DB2 11.1.


IBM Training

BLU declared global temporary tables

- Support for a column-organized DGTT
- Supports all options except not logged on rollback preserve rows
- Can be DB partitioned
- No BLU support for created global temporary tables

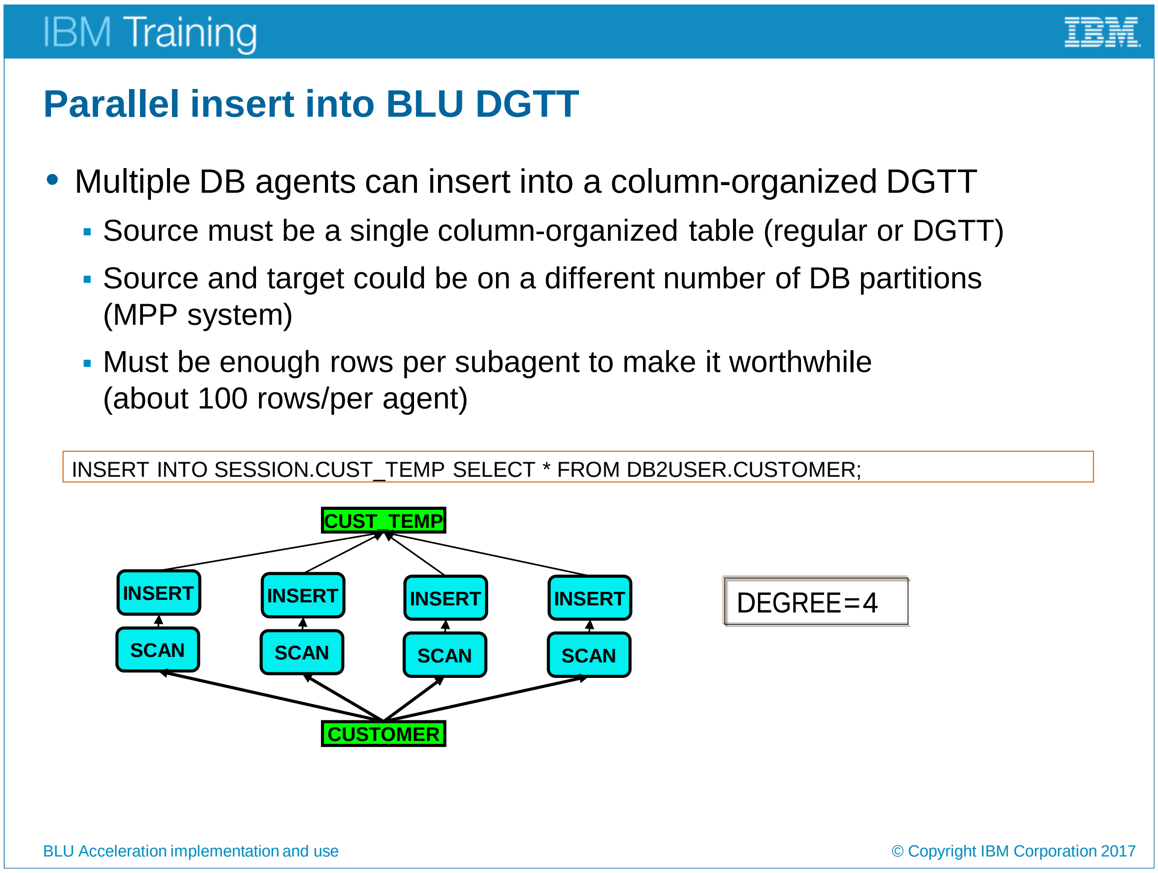
```
  DECLARE GLOBAL TEMPORARY TABLE
SESSION.CUST_TEMP (
  "C_CUSTOMER_SK" INTEGER NOT NULL ,
  "C_FIRST_NAME" CHAR(20 OCTETS) ,
  "C_LAST_NAME" CHAR(30 OCTETS) ,
  etc.
 )
 DISTRIBUTE BY HASH("C_CUSTOMER_SK") IN
USERTEMP1
 ORGANIZE BY COLUMN NOT LOGGED;
```

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

BLU declared global temporary tables

DB2 11.1 supports declared global temporary tables that are column-organized.

The slide shows an example of the statement an application could use to define a column-organized temporary table. This is supported in MPP partitioned databases as well as single partition databases.



IBM Training

Parallel insert into BLU DGTT

- Multiple DB agents can insert into a column-organized DGTT
  - Source must be a single column-organized table (regular or DGTT)
  - Source and target could be on a different number of DB partitions (MPP system)
  - Must be enough rows per subagent to make it worthwhile (about 100 rows/per agent)

```
INSERT INTO SESSION.CUST_TEMP SELECT * FROM DB2USER.CUSTOMER;
```

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Parallel insert into BLU DGTT

DB2 11.1 includes support for parallel insert processing for inserting large amounts of data into a column-organized temporary table.

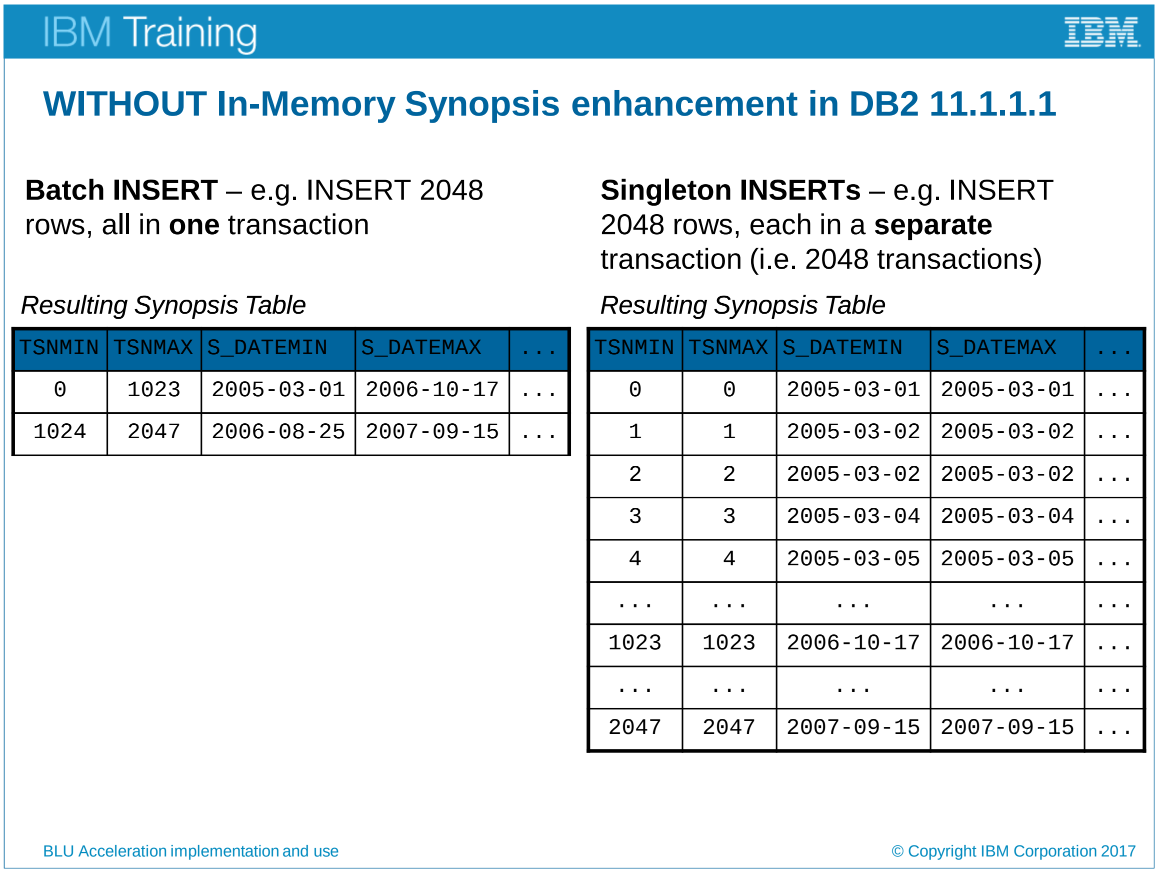
For example, consider this SQL statement:

```
        INSERT INTO SESSION.CUST_TEMP SELECT * FROM DB2USER.CUSTOMER
```

DB2 could utilize multiple DB2 agents to process the inserting of rows into the column-organized temporary table.

For parallel insert to be invoked, the source table would need to be a single column-organized table, but the source could be another declared global temporary table.

In a MPP database, the source and target tables do not need to be on the same database partitions.



IBM Training

WITHOUT In-Memory Synopsis enhancement in DB2 11.1.1.1

Batch INSERT – e.g. INSERT 2048 rows, all in **one** transaction

Resulting Synopsis Table

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

Singleton INSERTs – e.g. INSERT 2048 rows, each in a **separate** transaction (i.e. 2048 transactions)

Resulting Synopsis Table

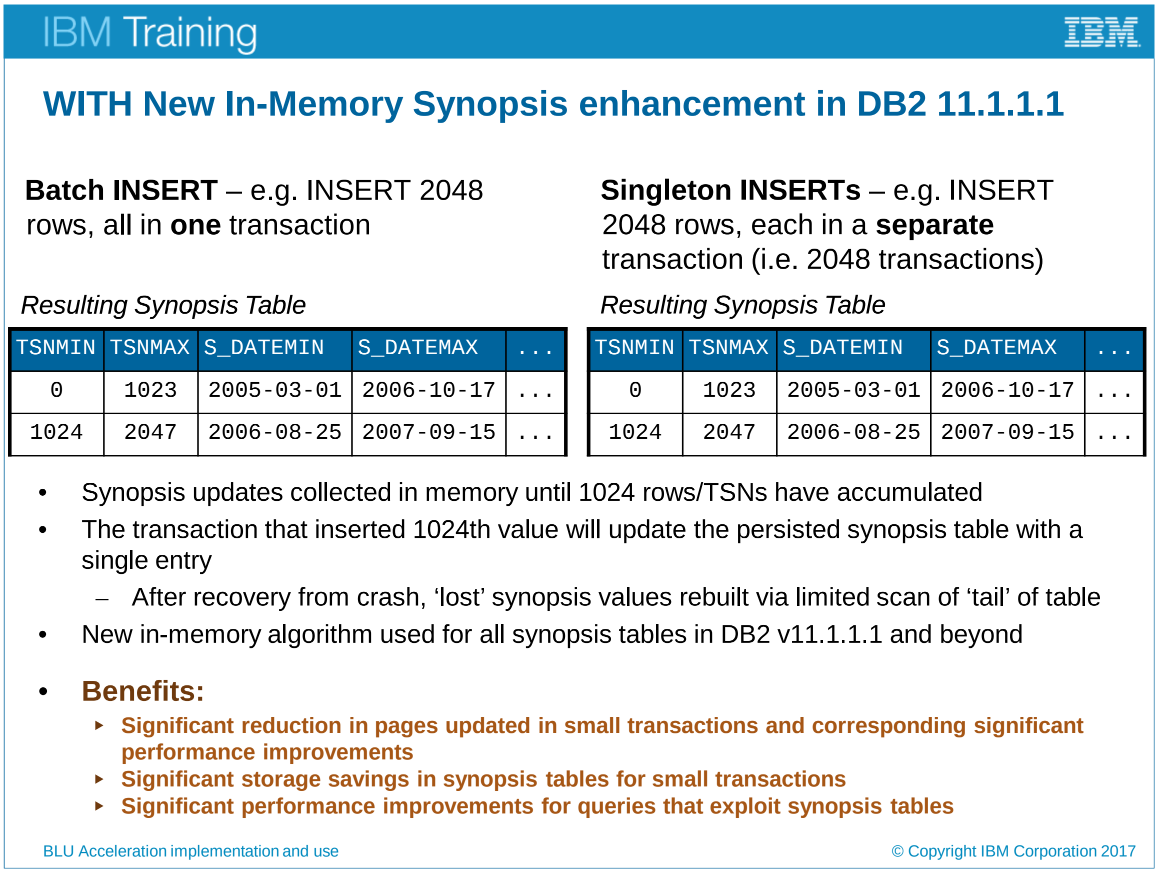
TSNMIN	TSNMAX	S_DATEMIN	S_DATEMAX	...
0	0	2005-03-01	2005-03-01	...
1	1	2005-03-02	2005-03-02	...
2	2	2005-03-02	2005-03-02	...
3	3	2005-03-04	2005-03-04	...
4	4	2005-03-05	2005-03-05	...
...	...	...	...	...
1023	1023	2006-10-17	2006-10-17	...
...	...	...	...	...
2047	2047	2007-09-15	2007-09-15	...

BLU Acceleration implementation and use

© Copyright IBM Corporation 2017

WITHOUT In-Memory Synopsis enhancement in DB2 11.1.1.1

Prior to the DB2 11.1.1.1 level, the Column-organized table support in DB2 maintained the synopsis table contents at the transaction level. This is efficient if large numbers of rows are added to the column-organized table in the transaction. If rows are inserted into the column-organized table, one per transaction, the synopsis table would need to be updated for each row inserted, which would increase the size of the synopsis table and reduce the efficiency of access when queries are processed.



IBM Training

WITH New In-Memory Synopsis enhancement in DB2 11.1.1.1

Batch INSERT – e.g. INSERT 2048 rows, all in **one** transaction

Resulting Synopsis Table

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

Singleton INSERTs – e.g. INSERT 2048 rows, each in a **separate** transaction (i.e. 2048 transactions)

Resulting Synopsis Table

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

- Synopsis updates collected in memory until 1024 rows/TSNs have accumulated
- The transaction that inserted 1024th value will update the persisted synopsis table with a single entry
  - After recovery from crash, 'lost' synopsis values rebuilt via limited scan of 'tail' of table
- New in-memory algorithm used for all synopsis tables in DB2 v11.1.1.1 and beyond
- **Benefits:**
  - **Significant reduction in pages updated in small transactions and corresponding significant performance improvements**
  - **Significant storage savings in synopsis tables for small transactions**
  - **Significant performance improvements for queries that exploit synopsis tables**

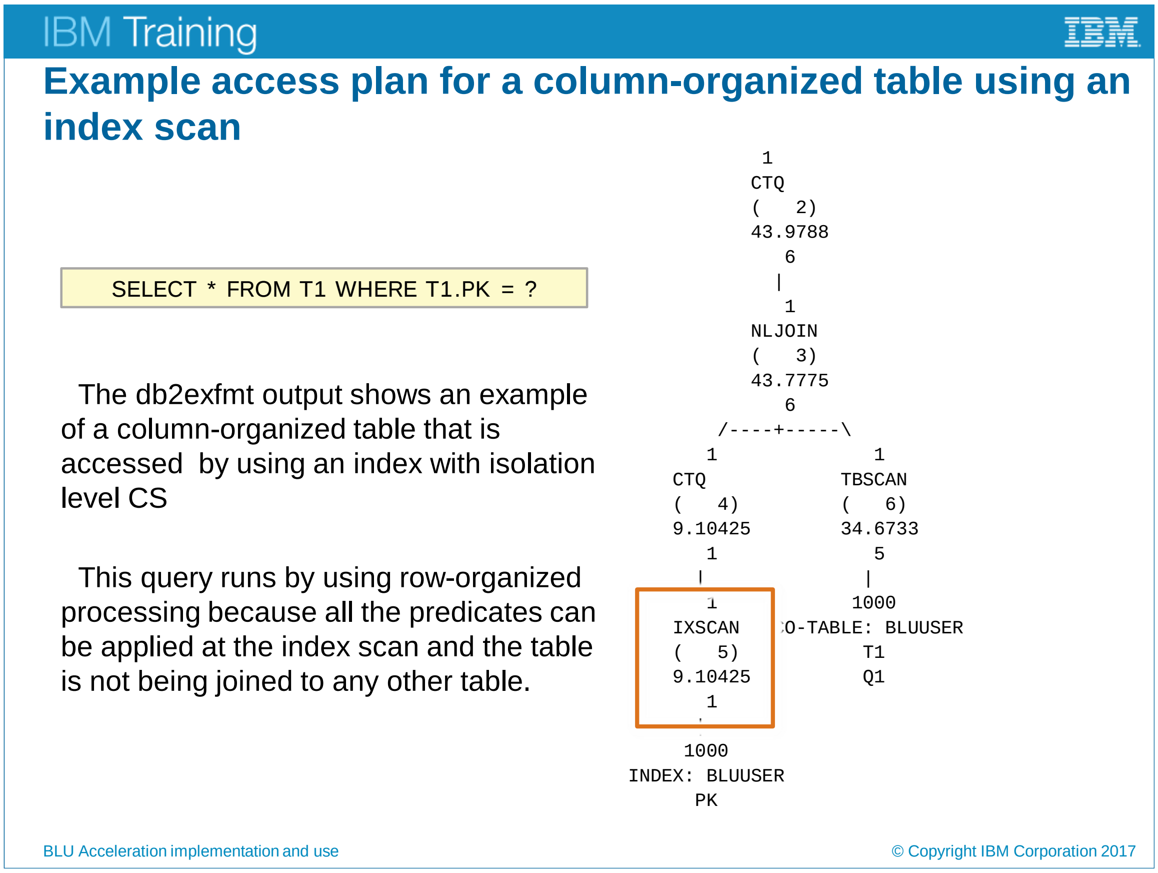
BLU Acceleration implementation and use © Copyright IBM Corporation 2017

WITH New In-Memory Synopsis enhancement in DB2 11.1.1.1

The Synopsis Table Maintenance improvements now in DB2 11.1.1.1 addresses the use case of transactions inserting a smaller set of rows into a column-organized table. This is the new behavior and automatic, by default.

Instead of updating the Synopsis Table either every 1024 rows or on transaction commit boundaries, DB2 will now cache data in memory (with algorithms to manage early DB shutdown or system crash scenarios) and only update the Synopsis Table every 1024 rows. This provides a significant performance improvement for transactions which insert a small number of rows which no longer need to update the Synopsis Table, with maximum improvements achieved with single-row insert transactions.

In addition to the insert performance, having fewer entries in the Synopsis Table not only allows the table to be smaller with fewer meta-data rows, but queries will also perform faster since there will be fewer rows to check and determine which set of column values to access in the column-organized table.


IBM Training

Example access plan for a column-organized table using an index scan

```
SELECT * FROM T1 WHERE T1.PK = ?
```

The db2exfmt output shows an example of a column-organized table that is accessed by using an index with isolation level CS

This query runs by using row-organized processing because all the predicates can be applied at the index scan and the table is not being joined to any other table.

```
                 1
                CTQ
                (   2)
                43.9788
                   6
                  |
                   1
                NLJOIN
                (   3)
                43.7775
                   6
             /----+-----\
          1                 1
       CTQ               TBSCAN
       (   4)            (   6)
       9.10425           34.6733
          1                 5
         |                 |
          1               1000
       IXSCAN        CO-TABLE: BLUUSER
       (   5)              T1
       9.10425             Q1
          1
         |
        1000
   INDEX: BLUUSER
          PK
```

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

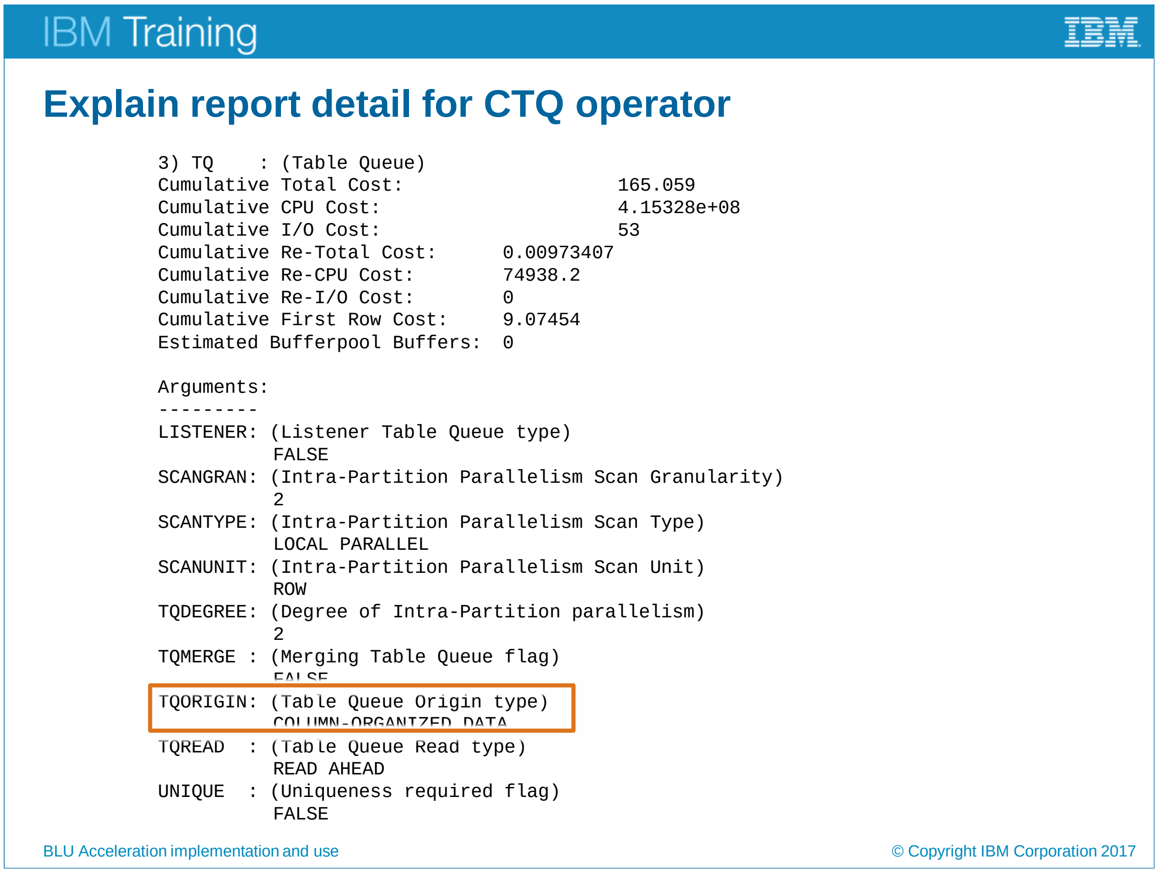
Example access plan for a column-organized table using an index scan

Starting with the DB2 Cancun Release 10.5.0.4 index access for SELECT statements that are run with isolation level CS (cursor stability) are supported.

An extra runtime optimization is available to improve the performance of column-organized UR and CS index scans when the data does not require column-organized processing. In some situations, it is more efficient for the access to be done by using row-organized processing because this approach avoids the overhead of switching between column-organized and row-organized formats. This additional optimization, in which index access is performed by using row-organized data processing, is not possible if all the following conditions apply:

- The index access occurs directly on the inner of a nested-loop join (NLJOIN).
- There are join predicates to be applied by the index scan.
- The join occurs between tables in the same subselect.

The db2exfmt output shows an example of a column-organized table that is accessed by using an index with isolation level CS. This query runs by using row-organized processing because all the predicates can be applied at the index scan and the table is not being joined to any other table.



IBM Training

Explain report detail for CTQ operator

```
3) TQ    : (Table Queue)
Cumulative Total Cost:                    165.059
Cumulative CPU Cost:                      4.15328e+08
Cumulative I/O Cost:                      53
Cumulative Re-Total Cost:      0.00973407
Cumulative Re-CPU Cost:        74938.2
Cumulative Re-I/O Cost:        0
Cumulative First Row Cost:     9.07454
Estimated Bufferpool Buffers:  0

Arguments:
---------
LISTENER: (Listener Table Queue type)
          FALSE
SCANGRAN: (Intra-Partition Parallelism Scan Granularity)
          2
SCANTYPE: (Intra-Partition Parallelism Scan Type)
          LOCAL PARALLEL
SCANUNIT: (Intra-Partition Parallelism Scan Unit)
          ROW
TQDEGREE: (Degree of Intra-Partition parallelism)
          2
TQMERGE : (Merging Table Queue flag)
          FALSE
TQORIGIN: (Table Queue Origin type)
          COLUMN-ORGANIZED DATA
TQREAD  : (Table Queue Read type)
          READ AHEAD
UNIQUE  : (Uniqueness required flag)
          FALSE
```

BLU Acceleration implementation and use

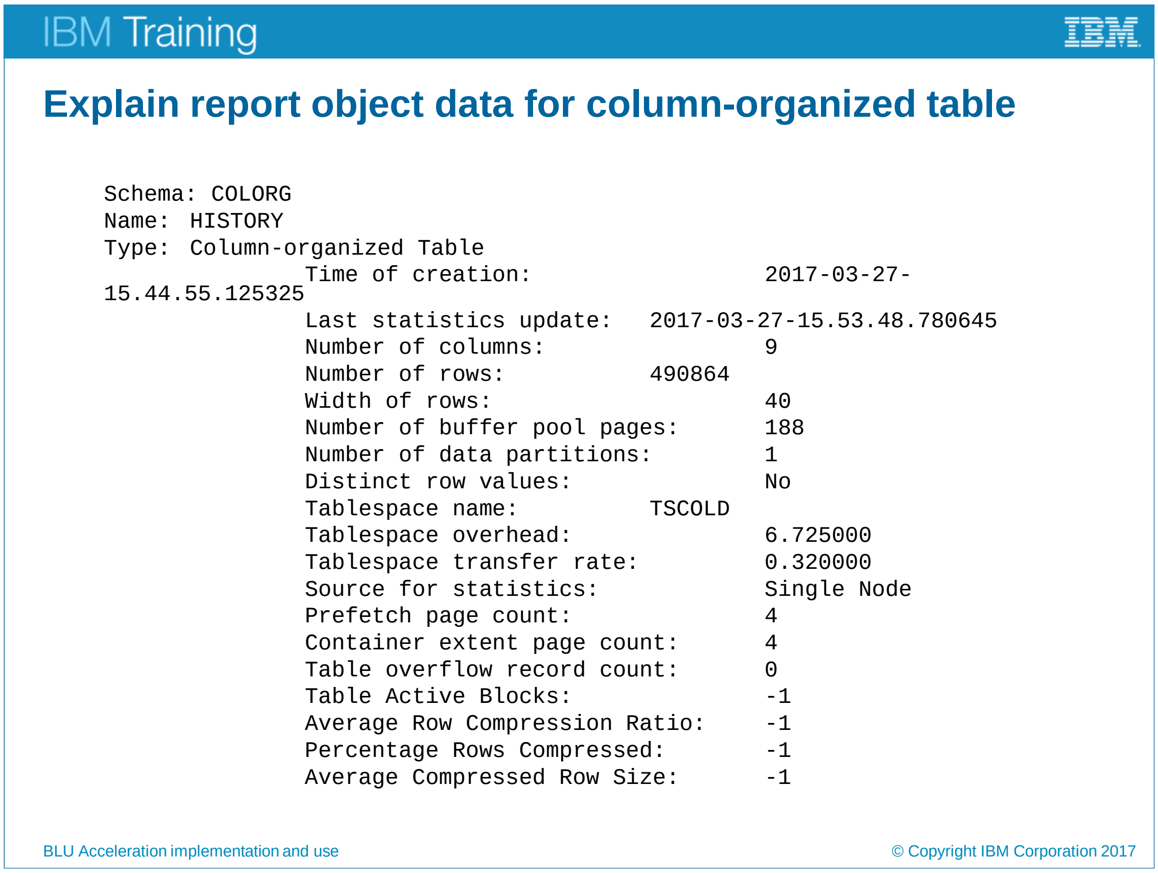
© Copyright IBM Corporation 2017

Explain report detail for CTQ operator

A table queue that is used to pass table data from one database agent to another.

The CTQ operator is a special type of table queue, used for processing queries with column-organized tables.

The sample detail section from a db2exfmt report contains this CTQ operator. The TQORIGIN in the Arguments section indicates COLUMN-ORGANIZED DATA.



IBM Training

Explain report object data for column-organized table

```
Schema: COLORG
Name:  HISTORY
Type:  Column-organized Table
               Time of creation:                 2017-03-27-
15.44.55.125325
               Last statistics update:   2017-03-27-15.53.48.780645
               Number of columns:                9
               Number of rows:           490864
               Width of rows:                    40
               Number of buffer pool pages:      188
               Number of data partitions:        1
               Distinct row values:              No
               Tablespace name:          TSCOLD
               Tablespace overhead:              6.725000
               Tablespace transfer rate:         0.320000
               Source for statistics:            Single Node
               Prefetch page count:              4
               Container extent page count:      4
               Table overflow record count:      0
               Table Active Blocks:              -1
               Average Row Compression Ratio:    -1
               Percentage Rows Compressed:       -1
               Average Compressed Row Size:      -1
```

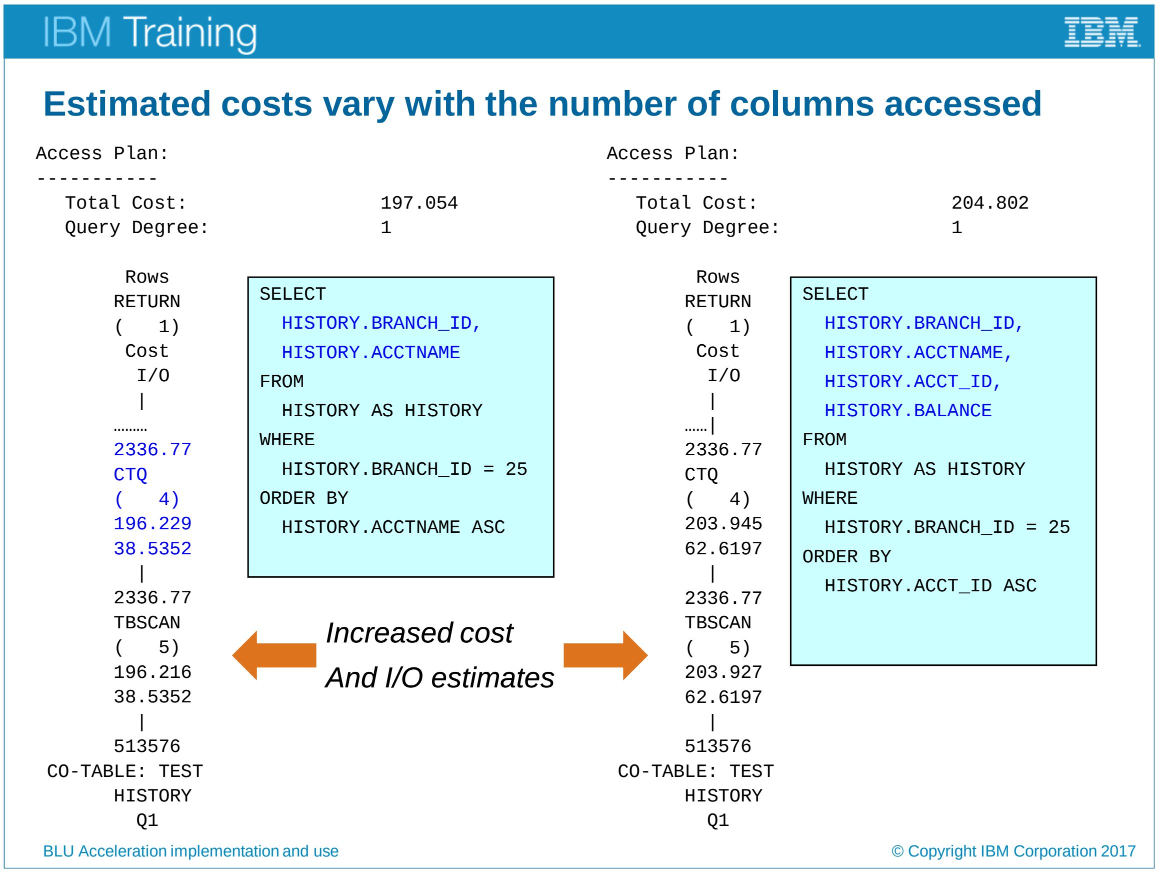
BLU Acceleration implementation and use

© Copyright IBM Corporation 2017

Explain report object data for column-organized table

Since a query may access some combination of row-organized and column-organized tables, in the objects used section of the db2exfmt report each column-organized table will show a type result value of: ‘Column-organized Table’.

The sample column-organized table object information shows some of the table statistics used by the DB2 optimizer to build and cost the access plan.



IBM Training

Estimated costs vary with the number of columns accessed

```
Access Plan:
-----------
  Total Cost:                197.054
  Query Degree:              1

         Rows
        RETURN
        (   1)
         Cost
          I/O
          |
        ………
        2336.77
        CTQ
        (   4)
        196.229
        38.5352
          |
        2336.77
        TBSCAN
        (   5)
        196.216
        38.5352
          |
        513576
 CO-TABLE: TEST
        HISTORY
          Q1
```

```
SELECT
  HISTORY.BRANCH_ID,
  HISTORY.ACCTNAME
FROM
  HISTORY AS HISTORY
WHERE
  HISTORY.BRANCH_ID = 25
ORDER BY
  HISTORY.ACCTNAME ASC
```

Increased cost

And I/O estimates

```
Access Plan:
-----------
  Total Cost:                204.802
  Query Degree:              1

         Rows
        RETURN
        (   1)
         Cost
          I/O
          |
        ……|
        2336.77
        CTQ
        (   4)
        203.945
        62.6197
          |
        2336.77
        TBSCAN
        (   5)
        203.927
        62.6197
          |
        513576
 CO-TABLE: TEST
        HISTORY
          Q1
```

```
SELECT
  HISTORY.BRANCH_ID,
  HISTORY.ACCTNAME,
  HISTORY.ACCT_ID,
  HISTORY.BALANCE
FROM
  HISTORY AS HISTORY
WHERE
  HISTORY.BRANCH_ID = 25
ORDER BY
  HISTORY.ACCT_ID ASC
```

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

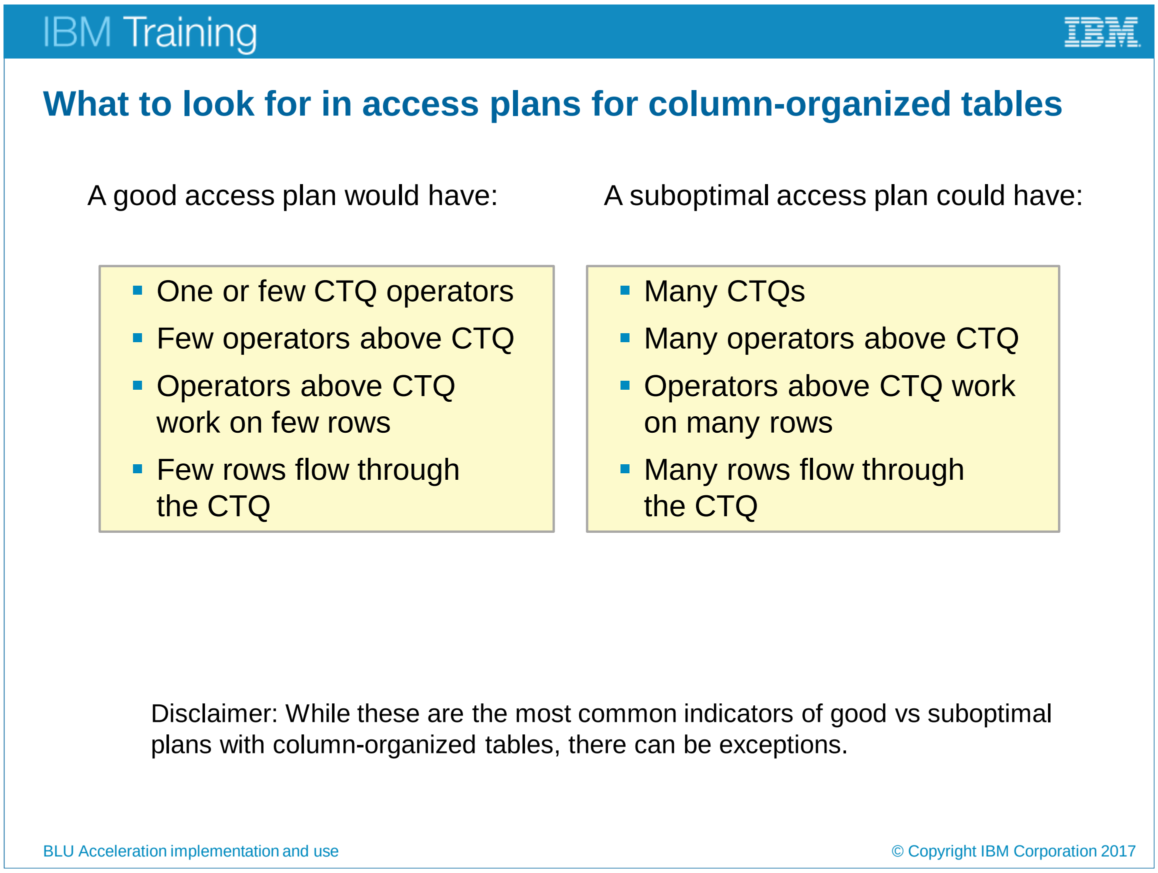
Estimated costs vary with the number of columns accessed

The slide shows two access plans from db2exfmt reports.

The SQL query text is similar for the two queries, except for the number of columns of data returned by each. The query on the left returns two columns of data, while the query on the right returns four columns of data.

Since the table used is a column-organized table, each column of data is stored in a distinct set of table pages. The example shows that the DB2 optimizer estimates additional I/O costs for the query that returns more columns in the result. The total estimated cost for the query that references more columns is slightly higher.

With column-organized tables it is especially important to only include the required columns in a query result to minimize costs and improve performance.



IBM Training

What to look for in access plans for column-organized tables

A good access plan would have:

- One or few CTQ operators
- Few operators above CTQ
- Operators above CTQ work on few rows
- Few rows flow through the CTQ

A suboptimal access plan could have:

- Many CTQs
- Many operators above CTQ
- Operators above CTQ work on many rows
- Many rows flow through the CTQ

Disclaimer: While these are the most common indicators of good vs suboptimal plans with column-organized tables, there can be exceptions.

BLU Acceleration implementation and use

© Copyright IBM Corporation 2017

What to look for in access plans for column-organized tables

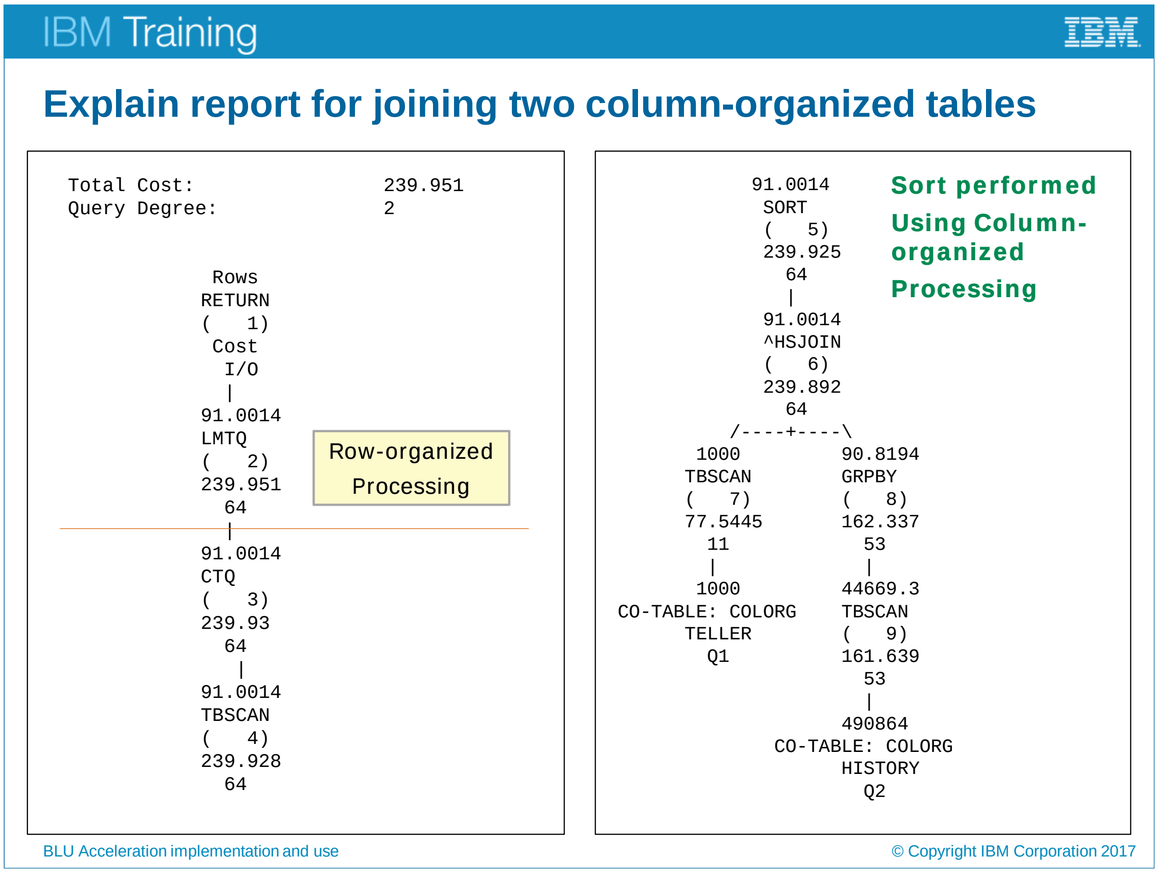
In many cases the performance characteristics of DB2 with BLU Acceleration will avoid the need to analyze SQL access plans. You are not concerned with index usage or join methods that you may need to analyze with row-organized tables.

Some general characteristics of a good access plan for column-organized tables are:

- One CTQ operator or possibly a few CTQ operators
- Few operators working using row-organized processing above the CTQ operator
- The operators above the CTQ are processing a small number of rows
- A small number of rows are flowing through the CTQ operator from column-organized processing into row-organized processing

Some characteristics of a suboptimal access plan for column-organized tables are:

- Many CTQ operators
- Many operators working using row-organized processing above the CTQ operator
- The operators above the CTQ are processing a large number of rows
- A large number of rows are flowing through the CTQ operator from column-organized processing into row-organized processing



IBM Training

Explain report for joining two column-organized tables

```
Total Cost:                 239.951
Query Degree:               2

            Rows
           RETURN
           (   1)
            Cost
             I/O
             |
           91.0014
           LMTQ
           (   2)
           239.951
             64
             |
           91.0014
           CTQ
           (   3)
           239.93
             64
             |
           91.0014
           TBSCAN
           (   4)
           239.928
             64
```

Row-organized Processing

```
                    91.0014
                     SORT
                     (   5)
                     239.925
                       64
                       |
                     91.0014
                     ^HSJOIN
                     (   6)
                     239.892
                       64
                  /----+----\
          1000            90.8194
         TBSCAN           GRPBY
         (   7)           (   8)
         77.5445          162.337
           11               53
           |                |
          1000            44669.3
  CO-TABLE: COLORG        TBSCAN
         TELLER           (   9)
           Q1             161.639
                            53
                            |
                          490864
                  CO-TABLE: COLORG
                          HISTORY
                            Q2
```

Sort performed

Using Column-organized Processing

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Explain report for joining two column-organized tables

The slide shows a portion of the access plan from an explain tool report for a query that joins two column-organized tables.

The SQL statement use was the following:

```
SELECT
  HISTORY.TELLER_ID,
  sum(HISTORY.BALANCE) as total_balance,
  TELLER.TELLER_NAME,
  count(*) as transactions
FROM
  COLORG.HISTORY AS HISTORY,
  COLORG.TELLER AS TELLER
WHERE
  HISTORY.TELLER_ID = TELLER.TELLER_ID and
  HISTORY.teller_id between 10 and 100
GROUP BY
  HISTORY.TELLER_ID,
  TELLER.TELLER_NAME
order by
  4 desc
```

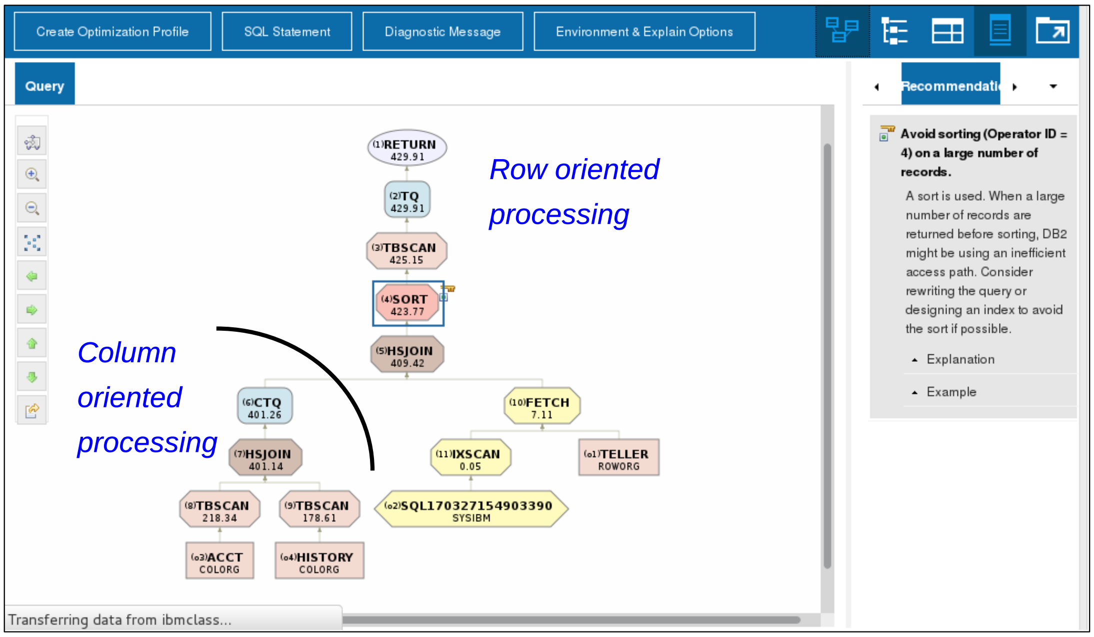
Notice that the two tables are joined using a HASH join, the HSJOIN operator in the column-organized processing section of the access plan. In most cases DB2 will utilize the smaller result set as the inner, right side, of the HASH join to reduce memory requirements.

Since HASH joins use DB2 sort memory, processing and performance costs can be impacted by insufficient sort memory for the database.

Using DB2 11, the SORT operation is performed in the column-organized engine before the CTQ operation.

IBM Training

DSM Explain tool joining two column-organized tables and one row-organized table



Avoid sorting (Operator ID = 4) on a large number of records.

A sort is used. When a large number of records are returned before sorting, DB2 might be using an inefficient access path. Consider rewriting the query or designing an index to avoid the sort if possible.

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

DSM Explain tool joining two column-organized tables and one row-organized table

For some applications, you may decide to use a mixture of column-organized and row-organized tables.

The slide shows an example EXPLAIN function using Data Server Manager, joining two column-organized tables with a third row-organized table.

Notice that DB2 joins the two column-organized tables using column-organized processing before the CTQ operator in the access plan.

The join operation with the row-organized must be performed using row-organized processing, above the CTQ operator in the access plan.

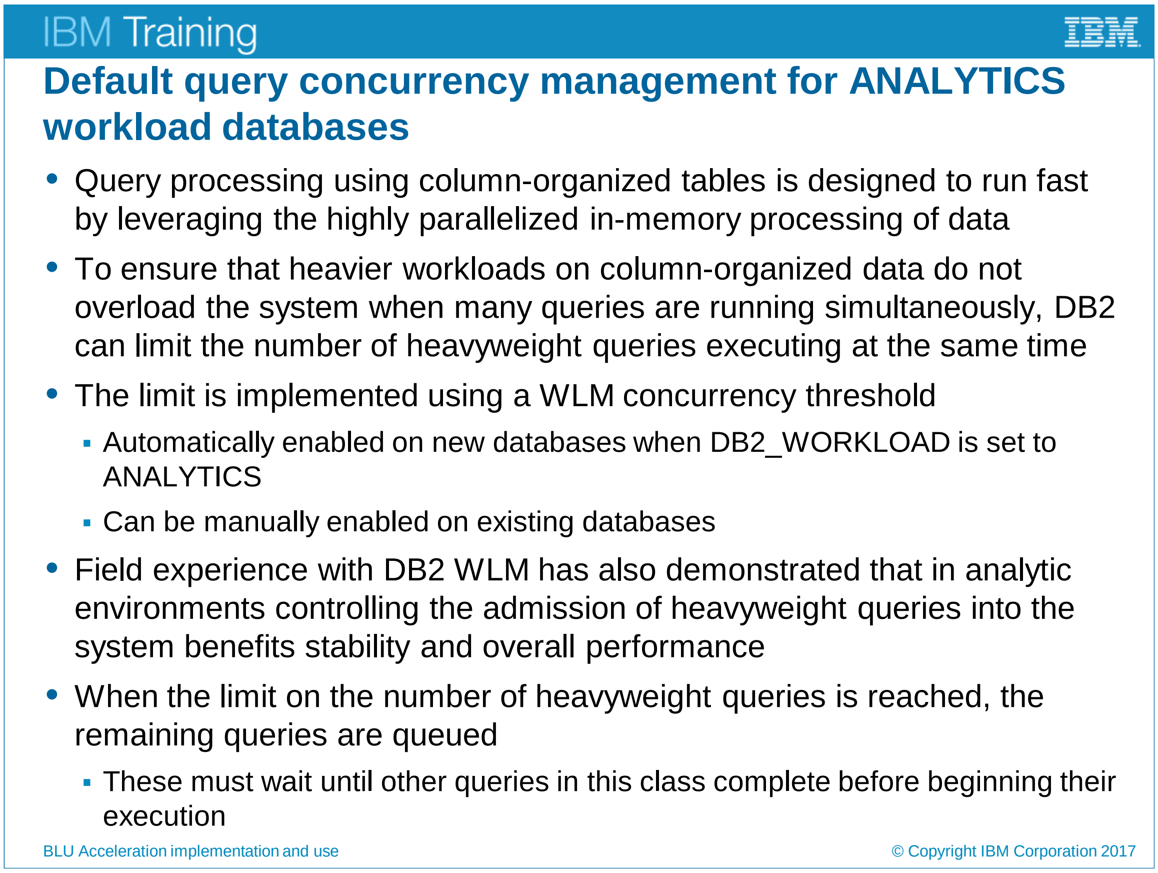
Instructor notes:

Purpose — To discuss an example db2expln explain report for a query joining three column-organized tables.

Details —

Additional information —

Transition statement — Next we will look at an example of an explain report joining column-organized and row-organized tables.



IBM Training

Default query concurrency management for ANALYTICS workload databases

- Query processing using column-organized tables is designed to run fast by leveraging the highly parallelized in-memory processing of data
- To ensure that heavier workloads on column-organized data do not overload the system when many queries are running simultaneously, DB2 can limit the number of heavyweight queries executing at the same time
- The limit is implemented using a WLM concurrency threshold
  - Automatically enabled on new databases when DB2_WORKLOAD is set to ANALYTICS
  - Can be manually enabled on existing databases
- Field experience with DB2 WLM has also demonstrated that in analytic environments controlling the admission of heavyweight queries into the system benefits stability and overall performance
- When the limit on the number of heavyweight queries is reached, the remaining queries are queued
  - These must wait until other queries in this class complete before beginning their execution

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Default query concurrency management for ANALYTICS workload databases

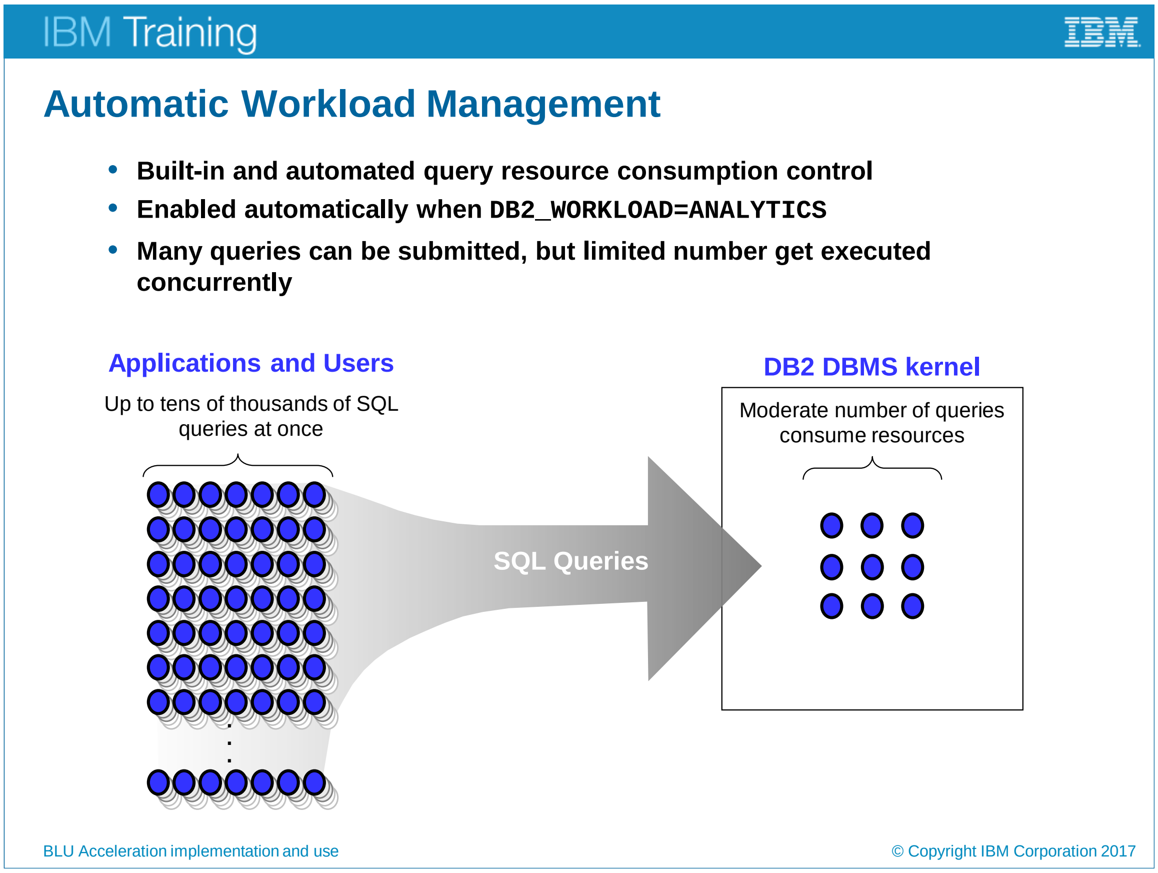
To ensure that heavier workloads on column-organized data do not overload the system when many queries are submitted simultaneously, there is a limit on the number of heavyweight queries that can execute on the database at the same time.

This limit can be implemented by using the default workload management concurrency threshold that is automatically enabled on new databases when the value of the DB2_WORKLOAD registry variable is set to ANALYTICS, or that can be manually enabled on existing databases.

The processing of queries against column-organized tables is designed to run fast by leveraging the highly parallelized in-memory processing of data. The trade-off for this high performance is that queries referencing column-organized tables also have a relatively large footprint in terms of memory and CPU when compared to similar queries processing row-organized table data. As such, the execution of these types of queries is optimal when relatively few of them are admitted to the system at a time. This enables them to individually leverage more processing power and memory on the system, and minimizes contention on the processor caches.

Field experience with DB2 workload management has also demonstrated that in analytic environments that support mixed workloads (where queries might vary widely in their degree of complexity and resource needs), controlling the admission of "heavyweight" queries into the system yields improvements in both system stability and overall performance, because resource overload on the system is avoided.

When the limit on the number of heavyweight queries is reached, the remaining queries are queued and must wait until other queries leave the system before beginning their execution. This can help to ensure system stability when a large number of complex ad hoc queries are running on systems that have not implemented a specific workload management strategy. Users who want to further optimize the execution of mixed workloads on their systems are encouraged to look at the full range of workload management capabilities offered in the DB2 database product.



IBM Training

Automatic Workload Management

- **Built-in and automated query resource consumption control**
- **Enabled automatically when `DB2_WORKLOAD=ANALYTICS`**
- **Many queries can be submitted, but limited number get executed concurrently**

Applications and Users
Up to tens of thousands of SQL queries at once
SQL Queries
DB2 DBMS kernel
Moderate number of queries consume resources

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

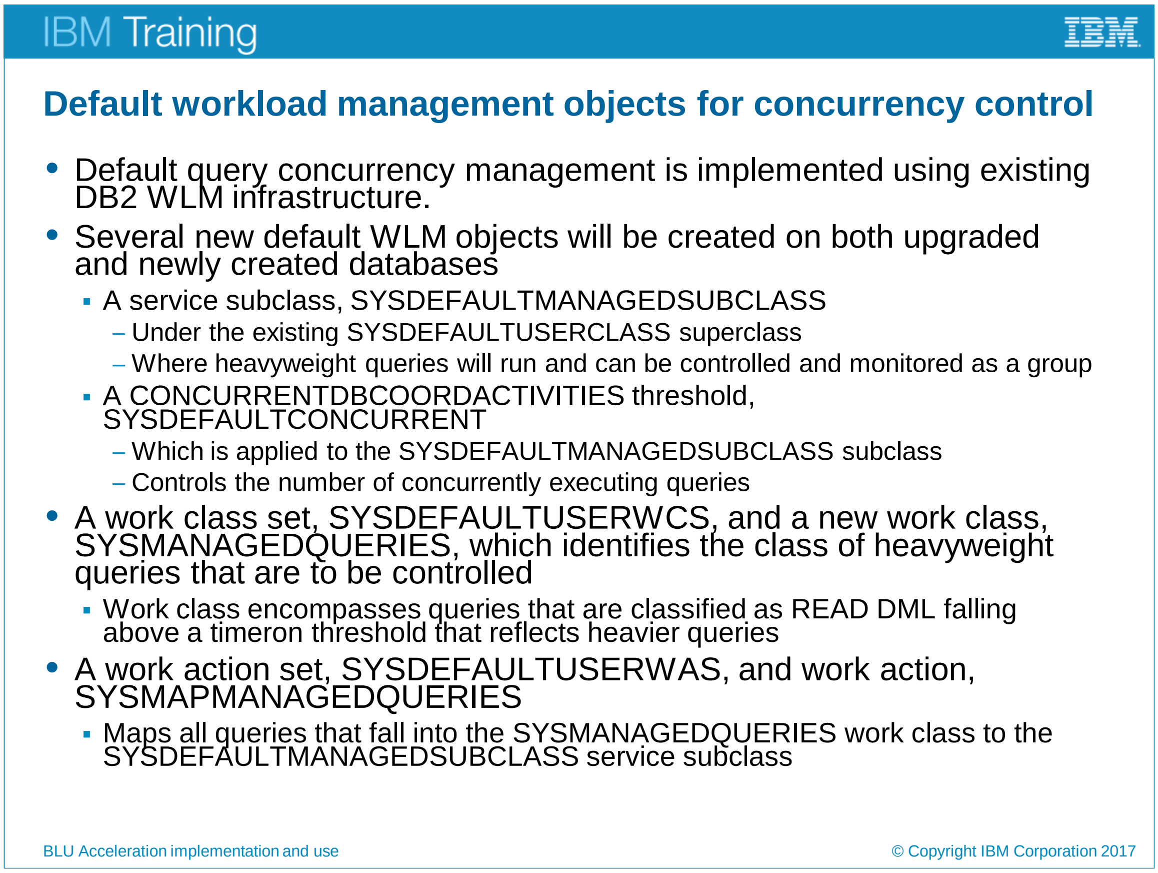
Automatic Workload Management

Starting with DB2 10.5, DB2 has built-in and automated query resource consumption controls.

Every additional query that runs concurrently naturally consumes more memory, locks, CPU, and memory bandwidth.

The DB2 BLU Acceleration feature automatically allows a high level of concurrent queries to be submitted for processing, but limits the number that consume resources at any point in time.

That means more memory and CPU for each query that is actively running. This benefits the entire analytics workload.



IBM Training

Default workload management objects for concurrency control

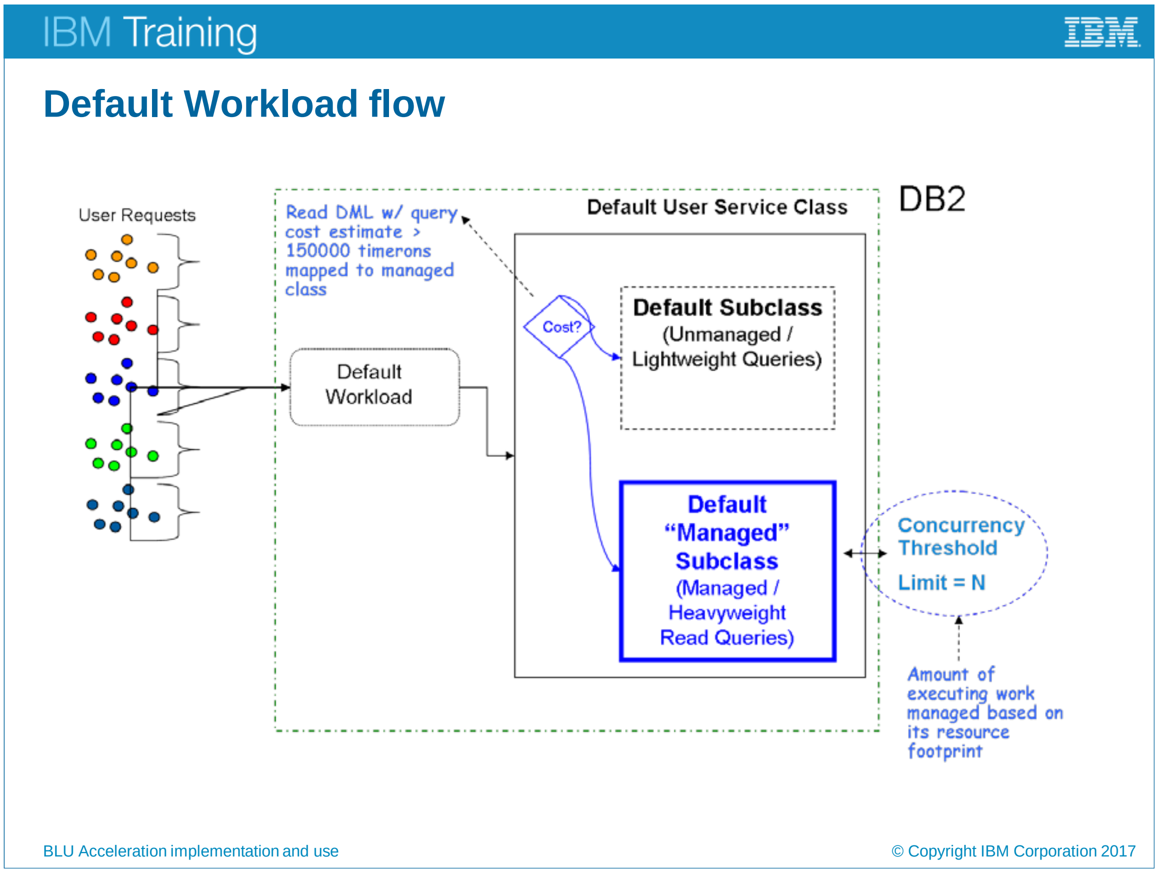
- Default query concurrency management is implemented using existing DB2 WLM infrastructure.
- Several new default WLM objects will be created on both upgraded and newly created databases
  - A service subclass, SYSDEFAULTMANAGEDSUBCLASS
    - Under the existing SYSDEFAULTUSERCLASS superclass
    - Where heavyweight queries will run and can be controlled and monitored as a group
  - A CONCURRENTDBCOORDACTIVITIES threshold, SYSDEFAULTCONCURRENT
    - Which is applied to the SYSDEFAULTMANAGEDSUBCLASS subclass
    - Controls the number of concurrently executing queries
- A work class set, SYSDEFAULTUSERWCS, and a new work class, SYSMANAGEDQUERIES, which identifies the class of heavyweight queries that are to be controlled
  - Work class encompasses queries that are classified as READ DML falling above a timeron threshold that reflects heavier queries
- A work action set, SYSDEFAULTUSERWAS, and work action, SYSMAPMANAGEDQUERIES
  - Maps all queries that fall into the SYSMANAGEDQUERIES work class to the SYSDEFAULTMANAGEDSUBCLASS service subclass

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Default workload management objects for concurrency control

Default query concurrency management is implemented by leveraging the existing DB2 workload management infrastructure.
Several new default workload management objects will be created on both upgraded and newly created databases:

- A service subclass, SYSDEFAULTMANAGEDSUBCLASS, under the existing SYSDEFAULTUSERCLASS superclass, where heavyweight queries will run and can be controlled and monitored as a group
- A CONCURRENTDBCOORDACTIVITIES threshold, SYSDEFAULTCONCURRENT, which is applied to the SYSDEFAULTMANAGEDSUBCLASS subclass to control the number of concurrently executing queries that are running in that subclass
- A work class set, SYSDEFAULTUSERWCS, and a new work class, SYSMANAGEDQUERIES, which identify the class of heavyweight queries that are to be controlled. The SYSMANAGEDQUERIES work class encompasses queries that are classified as READ DML (an existing work type for work classes) falling above a timeron threshold that reflects heavier queries.
- A work action set, SYSDEFAULTUSERWAS, and work action, SYSMAPMANAGEDQUERIES, which map all queries that fall into the SYSMANAGEDQUERIES work class to the SYSDEFAULTMANAGEDSUBCLASS service subclass


IBM Training

Default Workload flow

User Requests
Read DML w/ query cost estimate > 150000 timerons mapped to managed class
Default User Service Class
DB2
Default Workload
Cost?
Default Subclass
(Unmanaged / Lightweight Queries)
Default "Managed" Subclass
(Managed / Heavyweight Read Queries)
Concurrency Threshold
Limit = N
Amount of executing work managed based on its resource footprint

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Default Workload flow

This slide shows the default workload management for BLU in more detail. Here are the specifics:

- We split statements submitted to the system into two categories; unmanaged and managed.
- Read-only queries with an estimated cost of > 150000 timerons are mapped to the managed class.

The result is that:

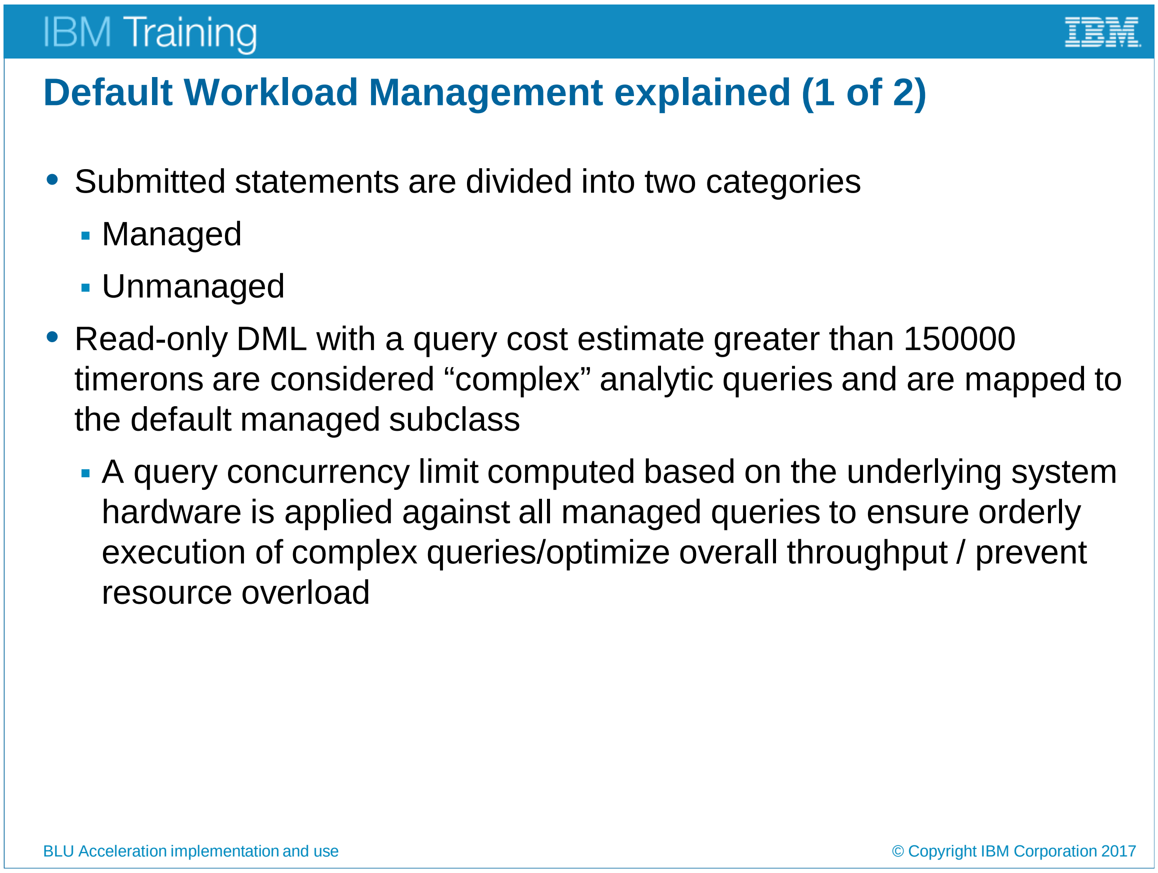
Heavy queries are queued and only N are executed concurrently allowing them to maximize their memory consumption / CPU parallelism and complete more quickly as well as preventing system overload.

We maintain the response time of lightweight point queries by allowing them to bypass the control and avoid queuing behind large queries; these queries have a much smaller resource impact on the system so we let them pass through as quickly as possible.

Other activities (DDL, Utilities, ETL) continue to be unmanaged. If managing these is desirable the WLM environment can be customized further.

We apply a concurrency limit to the managed class which is computed at database creation time based on the machine hardware and CPU parallelism to ensure orderly execution of heavier weight analytic queries.

Note that you can recompute this value if your system configuration changes by rerunning **AUTOCONFIGURE**.



IBM Training

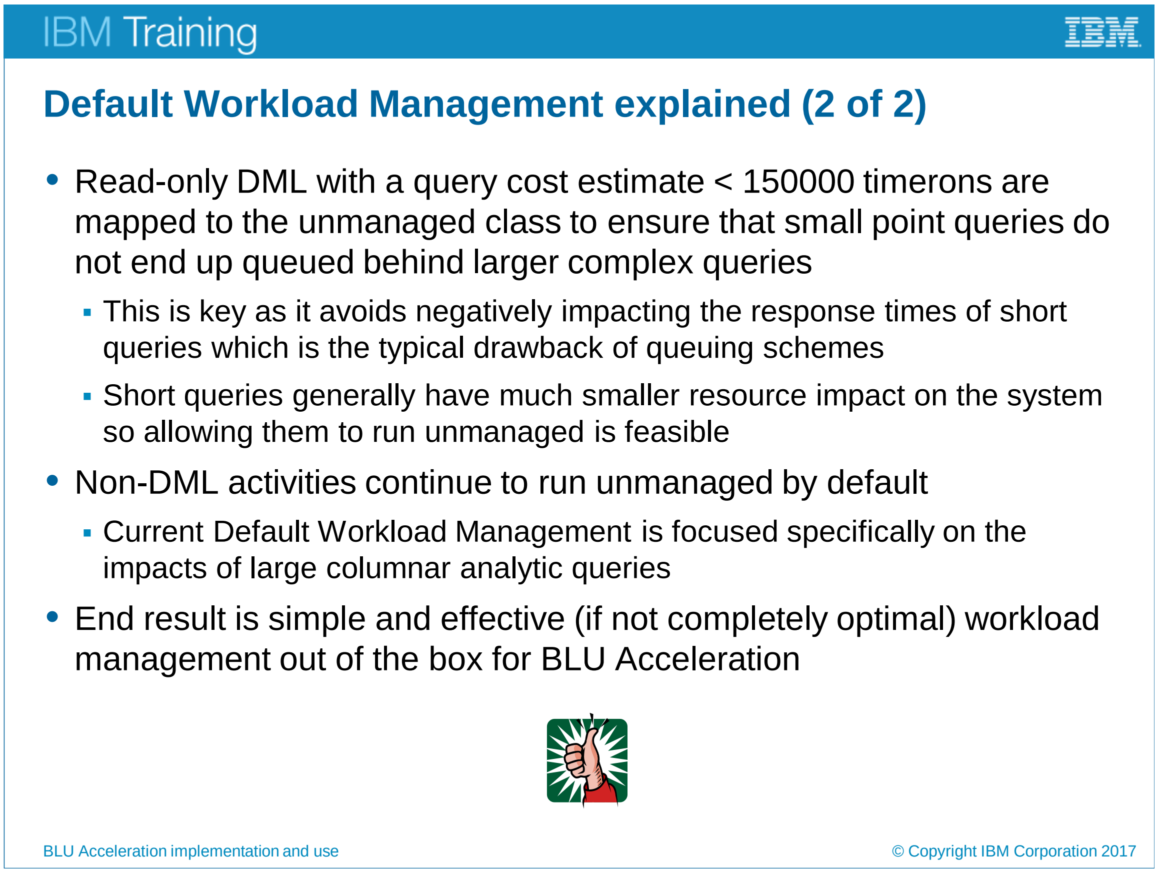
Default Workload Management explained (1 of 2)

- Submitted statements are divided into two categories
  - Managed
  - Unmanaged
- Read-only DML with a query cost estimate greater than 150000 timerons are considered “complex” analytic queries and are mapped to the default managed subclass
  - A query concurrency limit computed based on the underlying system hardware is applied against all managed queries to ensure orderly execution of complex queries/optimize overall throughput / prevent resource overload

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Default Workload Management explained

The workload management objects used by DB2 use a fixed query cost estimate to determine if a SQL query should be run using the managed service subclass where a concurrency limit is applied.



IBM Training

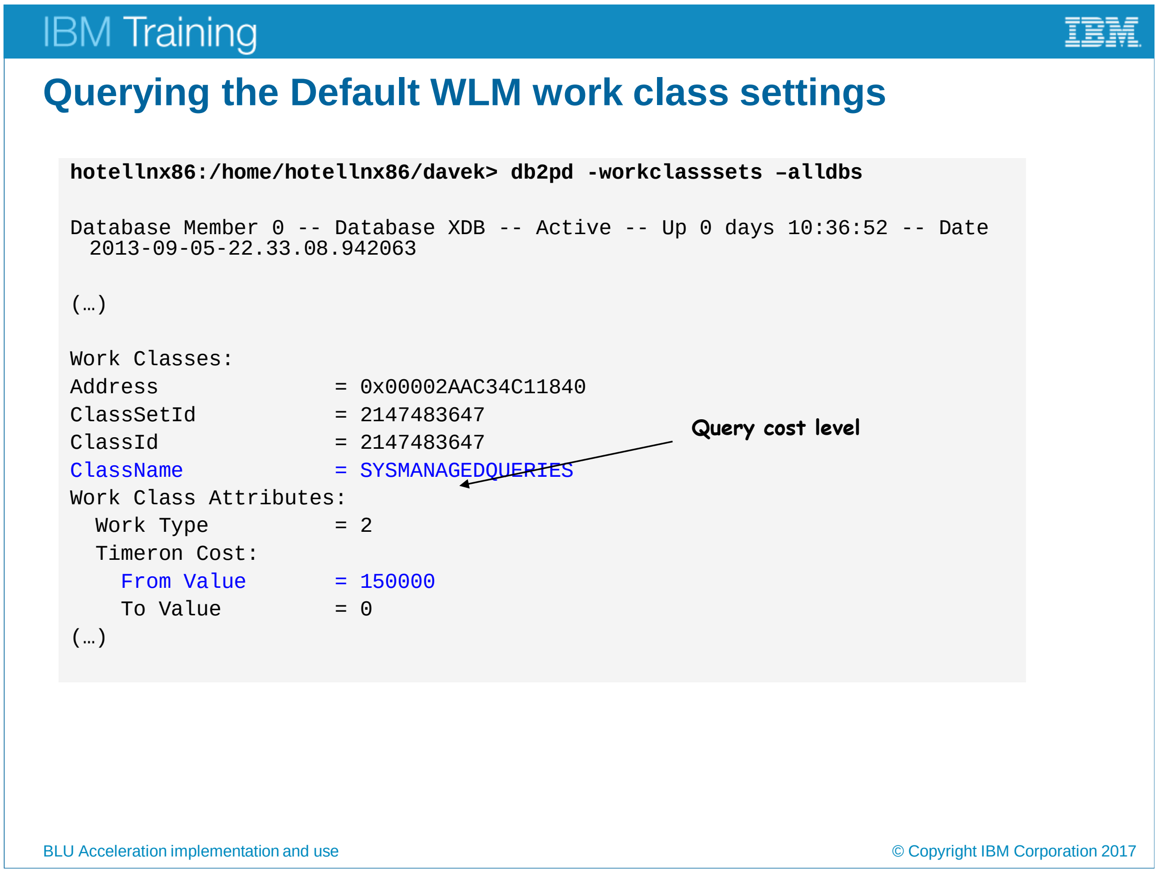
Default Workload Management explained (2 of 2)

- Read-only DML with a query cost estimate < 150000 timerons are mapped to the unmanaged class to ensure that small point queries do not end up queued behind larger complex queries
  - This is key as it avoids negatively impacting the response times of short queries which is the typical drawback of queuing schemes
  - Short queries generally have much smaller resource impact on the system so allowing them to run unmanaged is feasible
- Non-DML activities continue to run unmanaged by default
  - Current Default Workload Management is focused specifically on the impacts of large columnar analytic queries
- End result is simple and effective (if not completely optimal) workload management out of the box for BLU Acceleration

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

The end result is simple and effective workload management out of the box, resulting in more efficient use of resources, better stability, and better performance.

At the same time while this configuration is much better than an unmanaged system, it is a rather “blunt” instrument, and it would not be accurate to qualify it as optimal. In the next section we will explore how with a little bit of tuning you can further optimize these controls and really squeeze the best performance out of your system.



IBM Training

Querying the Default WLM work class settings

```
hotellnx86:/home/hotellnx86/davek> db2pd -workclasssets –alldbs

Database Member 0 -- Database XDB -- Active -- Up 0 days 10:36:52 -- Date
  2013-09-05-22.33.08.942063

(…)

Work Classes:
Address                   = 0x00002AAC34C11840
ClassSetId                = 2147483647
ClassId                   = 2147483647
ClassName                 = SYSMANAGEDQUERIES
Work Class Attributes:
  Work Type               = 2
  Timeron Cost:
    From Value            = 150000
    To Value              = 0
(…)
```

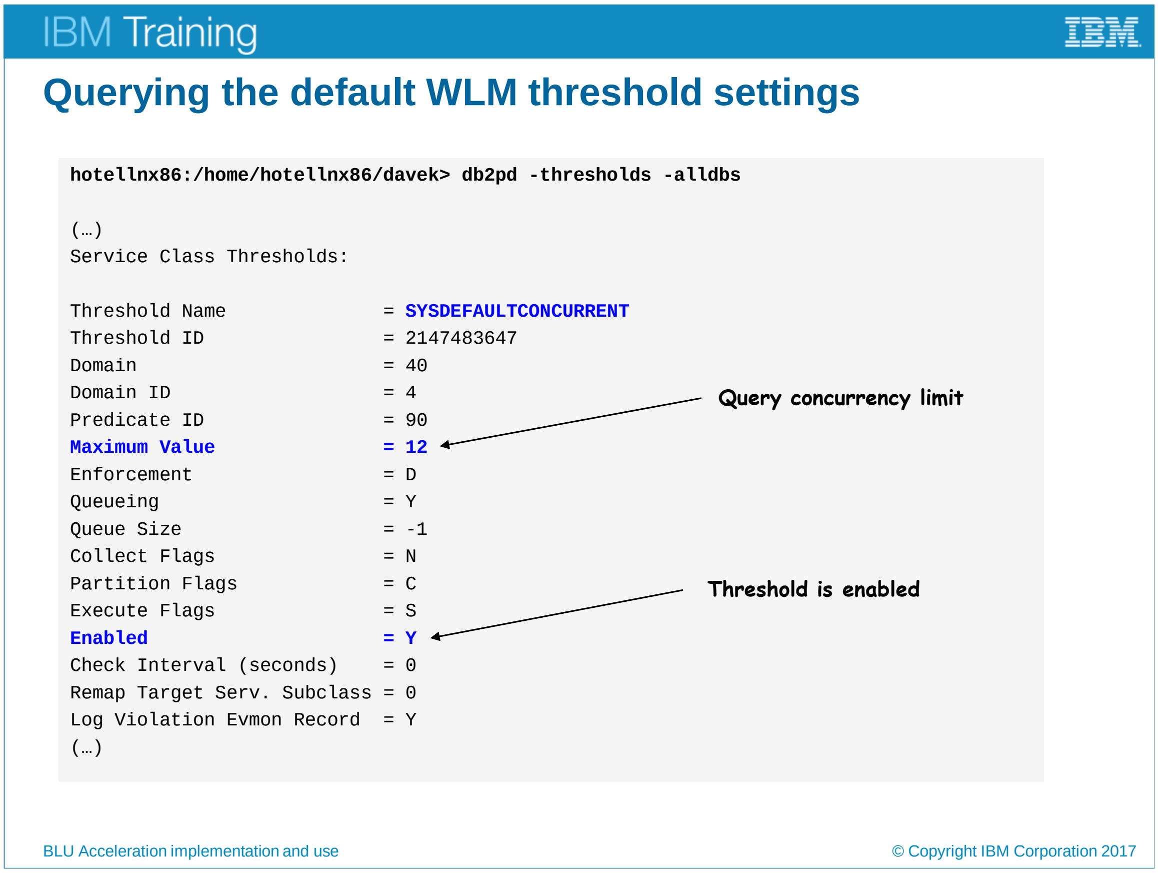
BLU Acceleration implementation and use

© Copyright IBM Corporation 2017

Querying the Default WLM work class settings

The **db2pd** command can be used to show the current setting for workload management objects in an active DB2 database.

The slide shows a portion of the report generated using the **-workclasssets** option of **db2pd**. The work class name is SYSMANAGEDQUERIES. The work class attributes show the lower threshold limit of 150000, with no upper limit.



IBM Training

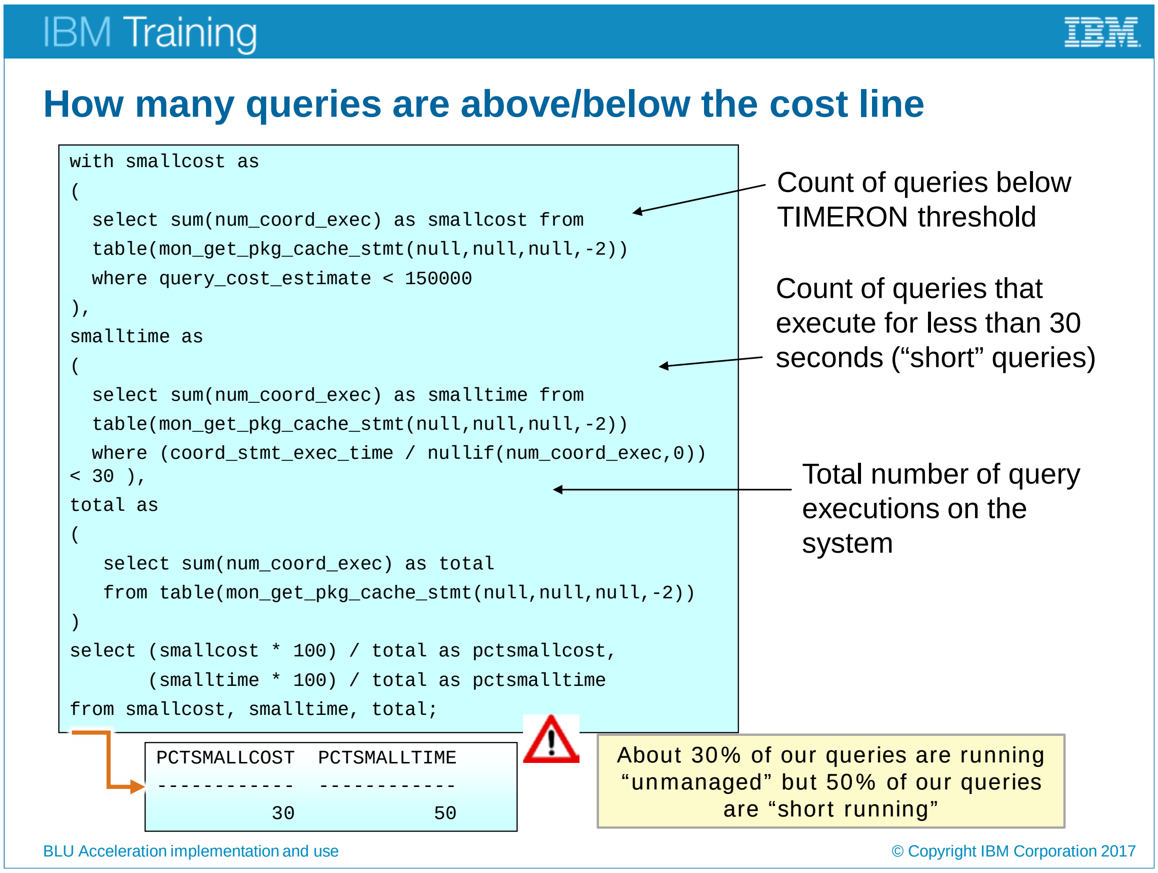
Querying the default WLM threshold settings

```
hotellnx86:/home/hotellnx86/davek> db2pd -thresholds -alldbs

(…)
Service Class Thresholds:

Threshold Name               = SYSDEFAULTCONCURRENT
Threshold ID                 = 2147483647
Domain                       = 40
Domain ID                    = 4
Predicate ID                 = 90
Maximum Value                = 12
Enforcement                  = D
Queueing                     = Y
Queue Size                   = -1
Collect Flags                = N
Partition Flags              = C
Execute Flags                = S
Enabled                      = Y
Check Interval (seconds)     = 0
Remap Target Serv. Subclass  = 0
Log Violation Evmon Record   = Y
(…)
```

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Querying the default WLM threshold settings

The slide shows a portion of the report generated using the **-thresholds** option of **db2pd**. The threshold name is SYSDEFAULTCONCURRENT. The report shows that the maximum concurrency is set to 12 on this system, and the threshold is enabled.



IBM Training

How many queries are above/below the cost line

```
with smallcost as
(
  select sum(num_coord_exec) as smallcost from
  table(mon_get_pkg_cache_stmt(null,null,null,-2))
  where query_cost_estimate < 150000
),
smalltime as
(
  select sum(num_coord_exec) as smalltime from
  table(mon_get_pkg_cache_stmt(null,null,null,-2))
  where (coord_stmt_exec_time / nullif(num_coord_exec,0))
< 30 ),
total as
(
   select sum(num_coord_exec) as total
   from table(mon_get_pkg_cache_stmt(null,null,null,-2))
)
select (smallcost * 100) / total as pctsmallcost,
       (smalltime * 100) / total as pctsmalltime
from smallcost, smalltime, total;
```

Count of queries below TIMERON threshold

Count of queries that execute for less than 30 seconds (“short” queries)

Total number of query executions on the system

```
PCTSMALLCOST  PCTSMALLTIME
------------  ------------
          30            50
```

About 30% of our queries are running “unmanaged” but 50% of our queries are “short running”

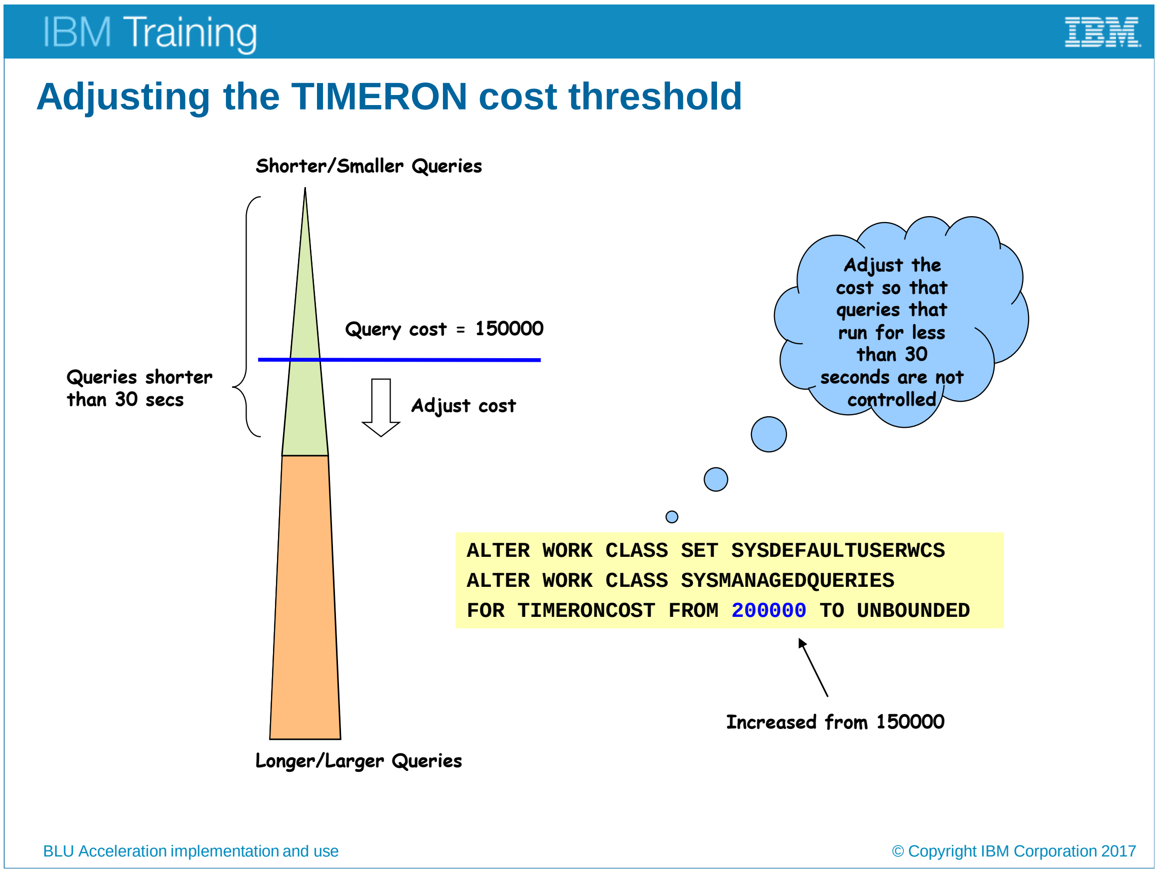
BLU Acceleration implementation and use

© Copyright IBM Corporation 2017

How many queries are above/below the cost line

The slide shows how the **MON_GET_PKG_CACHE_STMT** table function could be used to calculate a percentage of the SQL statements in the database package cache had an estimated cost that would cause them to be processed as unmanaged. In the example 30 percent of the statements in package cache would be unmanaged.

The query also calculates a percent of SQL statements that has an average execution time of 30 seconds or less. The sample result shows 50 percent of the statements had an average execution time of less than 30 seconds.


IBM Training
Shorter/Smaller Queries
Query cost = 150000
Queries shorter than 30 secs
Adjust cost
Adjust the cost so that queries that run for less than 30 seconds are not controlled
Increased from 150000
Longer/Larger Queries
BLU Acceleration implementation and use
© Copyright IBM Corporation 2017

Adjusting the TIMERON cost threshold

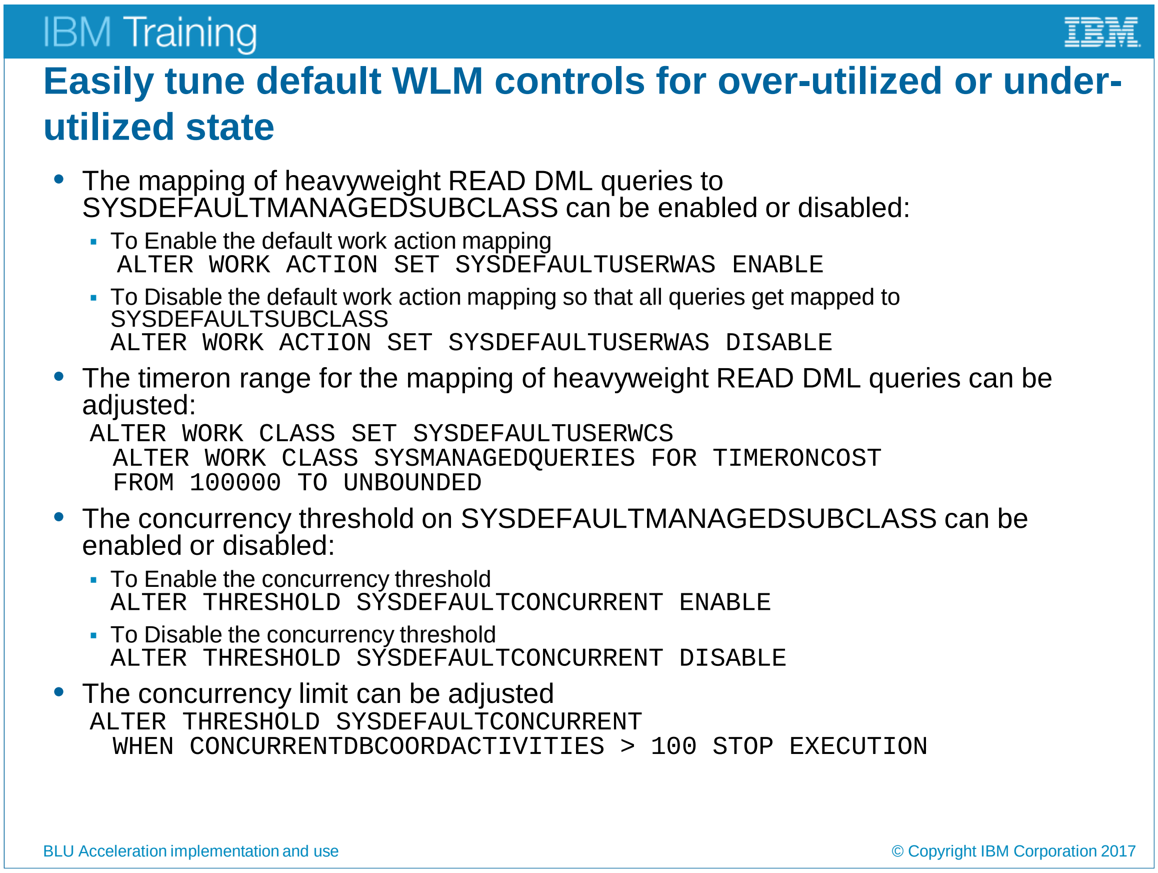
```
ALTER WORK CLASS SET SYSDEFAULTUSERWCS
ALTER WORK CLASS SYSMANAGEDQUERIES
FOR TIMERONCOST FROM 200000 TO UNBOUNDED
```

Adjusting the TIMERON cost threshold

This slide shows an illustration the spectrum of queries that have executed on our system, organized by execution time from shortest to longest. From our previous query we know that while approximately 30% of our queries are qualifying as unmanaged, roughly 50% fall into what we want to categorize as “short running”.

By iteratively increasing the query cost and rechecking our percentages we can tune our mapping criteria to ensure that all our “short running” queries are mapped to the unmanaged default subclass, helping to ensure we minimize their response times while still controlling the heavyweight queries submitted against the system.

Of course the inverse also applies and you may need to decrease the query cost of larger queries are getting mapped to the unmanaged class.


IBM Training

Easily tune default WLM controls for over-utilized or under-utilized state

- The mapping of heavyweight READ DML queries to SYSDEFAULTMANAGEDSUBCLASS can be enabled or disabled:
  - To Enable the default work action mapping
    ```
    ALTER WORK ACTION SET SYSDEFAULTUSERWAS ENABLE
    ```
  - To Disable the default work action mapping so that all queries get mapped to SYSDEFAULTSUBCLASS
    ```
    ALTER WORK ACTION SET SYSDEFAULTUSERWAS DISABLE
    ```
- The timeron range for the mapping of heavyweight READ DML queries can be adjusted:
  ```
  ALTER WORK CLASS SET SYSDEFAULTUSERWCS
    ALTER WORK CLASS SYSMANAGEDQUERIES FOR TIMERONCOST
    FROM 100000 TO UNBOUNDED
  ```
- The concurrency threshold on SYSDEFAULTMANAGEDSUBCLASS can be enabled or disabled:
  - To Enable the concurrency threshold
    ```
    ALTER THRESHOLD SYSDEFAULTCONCURRENT ENABLE
    ```
  - To Disable the concurrency threshold
    ```
    ALTER THRESHOLD SYSDEFAULTCONCURRENT DISABLE
    ```
- The concurrency limit can be adjusted
  ```
  ALTER THRESHOLD SYSDEFAULTCONCURRENT
    WHEN CONCURRENTDBCOORDACTIVITIES > 100 STOP EXECUTION
  ```

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Easily tune default WLM controls for over-utilized or under-utilized state

If the system appears to be running in an **under-utilized** state, take the following steps:

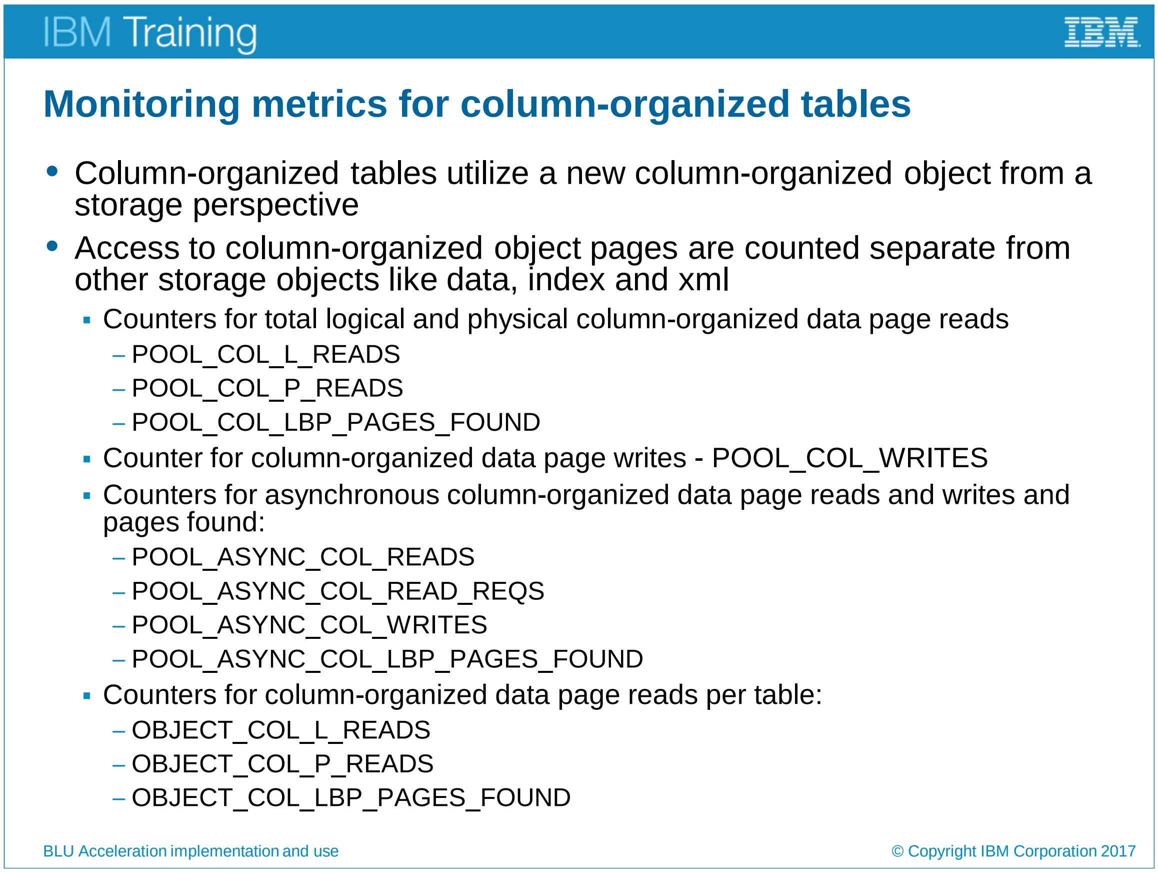
Examine the WLM_QUEUE_TIME_TOTAL metric, which is reported by various system-level table functions (or the statistics event monitor), to determine whether queries in the system are accumulating time waiting on concurrency thresholds.

- If no such queue time is being accumulated, the system is simply running under peak capacity, and no tuning is necessary.
- If queue time is being accumulated, work is being queued on the system. Monitor the amount of work running in SYSDEFAULTSUBCLASS and SYSDEFAULTMANAGEDSUBCLASS, and consider incrementally increasing the TIMERONCOST minimum on SYSMANAGEDQUERIES if it appears that too large a proportion of the workload is running within the managed class.

Assuming that the distribution of managed and unmanaged work appears reasonable, consider incrementally increasing the concurrency limit that is specified by SYSDEFAULTCONCURRENT until system resource usage reaches the target level.

If the system appears to be running in an **over-utilized** state, take the following steps:

- Monitor the amount of work running in SYSDEFAULTSUBCLASS and SYSDEFAULTMANAGEDSUBCLASS, and consider incrementally decreasing the TIMERONCOST minimum on SYSMANAGEDQUERIES if it appears that too small a proportion of the workload is running within the managed class.
- Assuming that the distribution of managed and unmanaged work appears reasonable, consider incrementally decreasing the concurrency limit that is specified by SYSDEFAULTCONCURRENT until system resource usage is back within the target range.


IBM Training

Monitoring metrics for column-organized tables

- Column-organized tables utilize a new column-organized object from a storage perspective
- Access to column-organized object pages are counted separate from other storage objects like data, index and xml
  - Counters for total logical and physical column-organized data page reads
    - POOL_COL_L_READS
    - POOL_COL_P_READS
    - POOL_COL_LBP_PAGES_FOUND
  - Counter for column-organized data page writes - POOL_COL_WRITES
  - Counters for asynchronous column-organized data page reads and writes and pages found:
    - POOL_ASYNC_COL_READS
    - POOL_ASYNC_COL_READ_REQS
    - POOL_ASYNC_COL_WRITES
    - POOL_ASYNC_COL_LBP_PAGES_FOUND
  - Counters for column-organized data page reads per table:
    - OBJECT_COL_L_READS
    - OBJECT_COL_P_READS
    - OBJECT_COL_LBP_PAGES_FOUND

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Monitoring metrics for column-organized tables

A set of monitor elements enables the monitoring of data page I/O for column-organized tables separately from that of row-organized tables. You can use these monitor elements to understand what portion of the I/O is being driven by access to column-organized tables when a workload impacts both row-organized and column-organized tables. These elements can also help you to tune the system, for example, by helping you to decide whether to place column-organized tables in separate table spaces, or whether to use a separate buffer pool. A column-organized data page contains column data for a column-organized table.

Counters for total logical and physical column-organized data page reads and pages found:

- POOL_COL_L_READS
- POOL_COL_P_READS
- POOL_COL_LBP_PAGES_FOUND

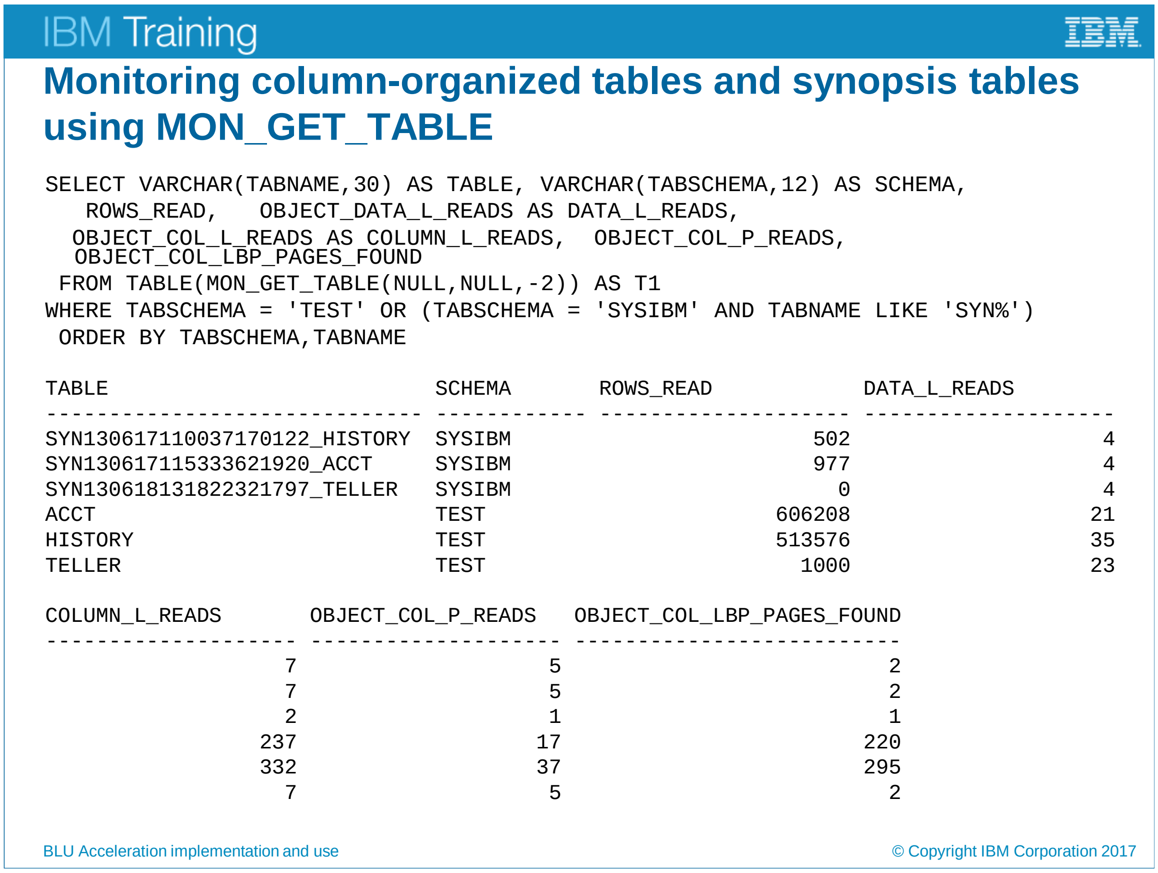
Counters for column-organized data page writes: POOL_COL_WRITES

Counters for asynchronous column-organized data page reads and writes and pages found:

- POOL_ASYNC_COL_READS
- POOL_ASYNC_COL_READ_REQS
- POOL_ASYNC_COL_WRITES
- POOL_ASYNC_COL_LBP_PAGES_FOUND

Counters for column-organized data page reads per table (and per statement per table, reported through monitor usage lists):

- OBJECT_COL_L_READS
- OBJECT_COL_P_READS
- OBJECT_COL_LBP_PAGES_FOUND



IBM Training

Monitoring column-organized tables and synopsis tables using MON_GET_TABLE

```
SELECT VARCHAR(TABNAME,30) AS TABLE, VARCHAR(TABSCHEMA,12) AS SCHEMA,
   ROWS_READ,   OBJECT_DATA_L_READS AS DATA_L_READS,
  OBJECT_COL_L_READS AS COLUMN_L_READS,  OBJECT_COL_P_READS,
  OBJECT_COL_LBP_PAGES_FOUND
 FROM TABLE(MON_GET_TABLE(NULL,NULL,-2)) AS T1
WHERE TABSCHEMA = 'TEST' OR (TABSCHEMA = 'SYSIBM' AND TABNAME LIKE 'SYN%')
 ORDER BY TABSCHEMA,TABNAME

TABLE                          SCHEMA       ROWS_READ            DATA_L_READS
------------------------------ ------------ -------------------- --------------------
SYN130617110037170122_HISTORY  SYSIBM                        502                    4
SYN130617115333621920_ACCT     SYSIBM                        977                    4
SYN130618131822321797_TELLER   SYSIBM                          0                    4
ACCT                           TEST                       606208                   21
HISTORY                        TEST                       513576                   35
TELLER                         TEST                         1000                   23

COLUMN_L_READS       OBJECT_COL_P_READS   OBJECT_COL_LBP_PAGES_FOUND
-------------------- -------------------- --------------------------
                   7                    5                          2
                   7                    5                          2
                   2                    1                          1
                 237                   17                        220
                 332                   37                        295
                   7                    5                          2
```

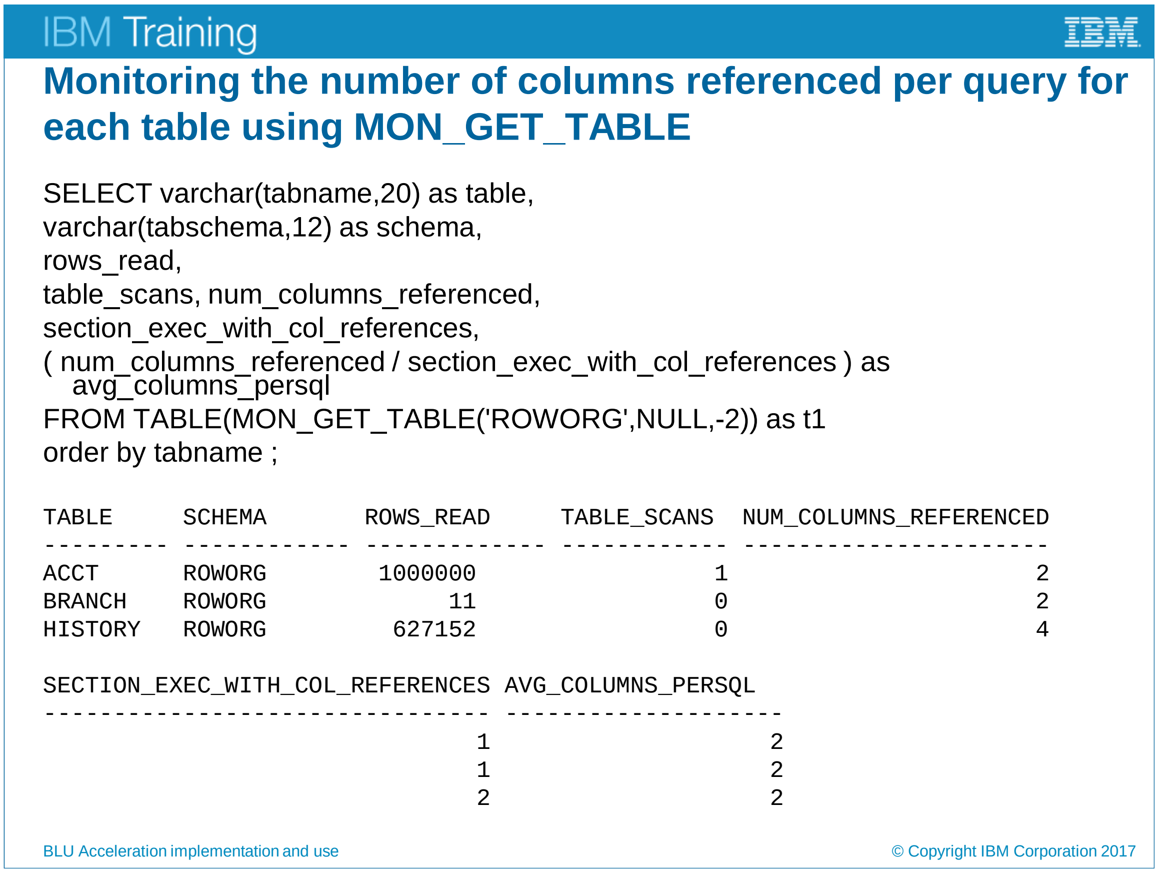
BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Monitoring column-organized tables and synopsis tables using MON_GET_TABLE

The example query uses the MON_GET_TABLE monitor function to return selected table activity statistics.

The query specifies a set of column-organized tables in the schema name ‘TEST’. There are standard statistics like ROWS_READ and several new monitor elements like OBJECT_COL_P_READS and OBJECT_COL_LBP_PAGES_FOUND showing processing for the pages in the column-organized object. Notice the counts for pages read from the table data object. These are references to the column dictionaries and other table metadata.

The query also includes the synopsis tables that DB2 creates and uses internally, to better understand the column-organized table processing. Since synopsis tables are internally managed column-organized tables, access to these will also be tracked with the same monitor elements.



IBM Training

Monitoring the number of columns referenced per query for each table using MON_GET_TABLE

```
SELECT varchar(tabname,20) as table,
varchar(tabschema,12) as schema,
rows_read,
table_scans, num_columns_referenced,
section_exec_with_col_references,
( num_columns_referenced / section_exec_with_col_references ) as
   avg_columns_persql
FROM TABLE(MON_GET_TABLE('ROWORG',NULL,-2)) as t1
order by tabname ;
```

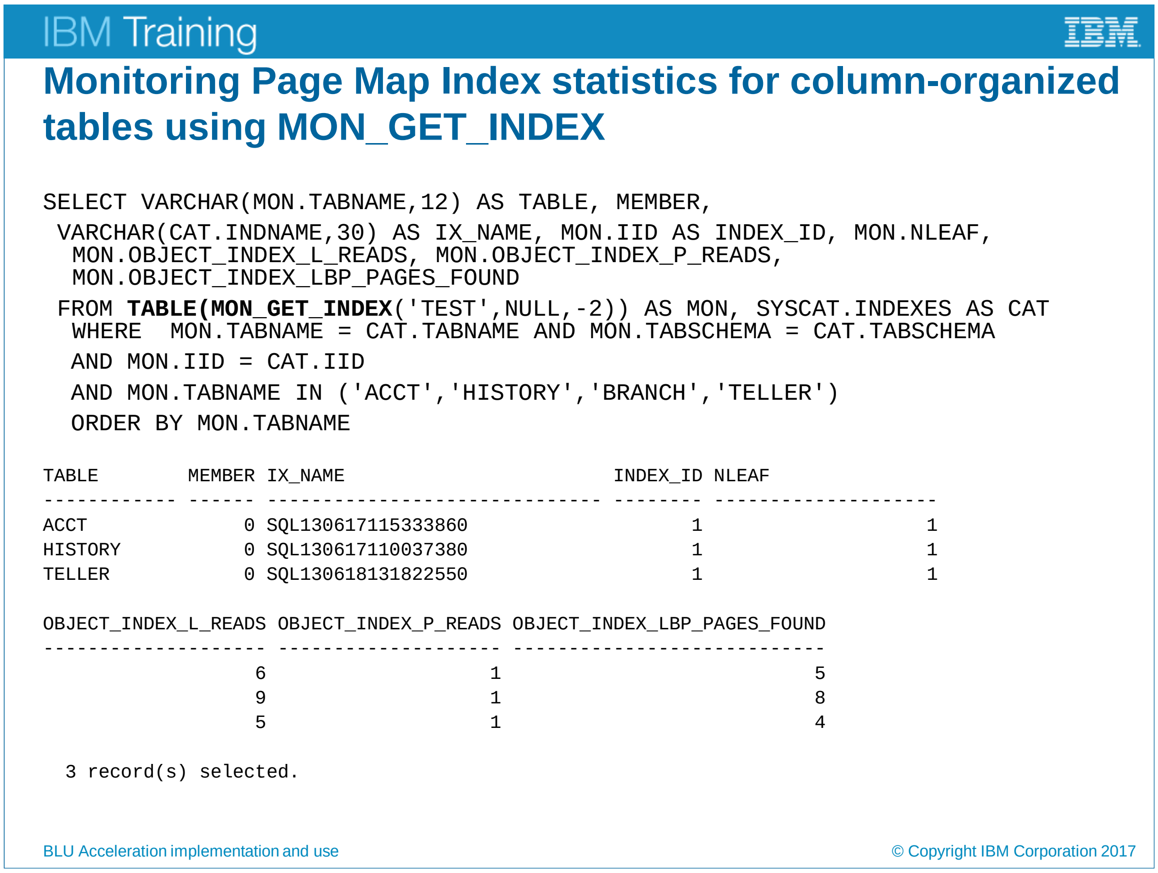
```
TABLE     SCHEMA       ROWS_READ     TABLE_SCANS  NUM_COLUMNS_REFERENCED
--------- ------------ ------------- ------------ ----------------------
ACCT      ROWORG         1000000                1                      2
BRANCH    ROWORG              11                0                      2
HISTORY   ROWORG          627152                0                      4

SECTION_EXEC_WITH_COL_REFERENCES AVG_COLUMNS_PERSQL
-------------------------------- --------------------
                               1                    2
                               1                    2
                               2                    2
```

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Monitoring the number of columns referenced per query for each table using MON_GET_TABLE

The elements SECTION_EXEC_WITH_COL_REFERENCES and NUM_COLUMNS_REFERENCED returned by MON_GET_TABLE can be used to determine the average number of columns being accessed from a table during execution of the runtime section for an SQL statement.

This average column access count can help identify row-organized tables that might be candidates for conversion to column-organized tables (for example, wide tables where only a few columns are typically accessed).


IBM Training

Monitoring Page Map Index statistics for column-organized tables using MON_GET_INDEX

```
SELECT VARCHAR(MON.TABNAME,12) AS TABLE, MEMBER,
 VARCHAR(CAT.INDNAME,30) AS IX_NAME, MON.IID AS INDEX_ID, MON.NLEAF,
  MON.OBJECT_INDEX_L_READS, MON.OBJECT_INDEX_P_READS,
  MON.OBJECT_INDEX_LBP_PAGES_FOUND
 FROM TABLE(MON_GET_INDEX('TEST',NULL,-2)) AS MON, SYSCAT.INDEXES AS CAT
  WHERE  MON.TABNAME = CAT.TABNAME AND MON.TABSCHEMA = CAT.TABSCHEMA
  AND MON.IID = CAT.IID
  AND MON.TABNAME IN ('ACCT','HISTORY','BRANCH','TELLER')
  ORDER BY MON.TABNAME

TABLE        MEMBER IX_NAME                        INDEX_ID NLEAF
------------ ------ ------------------------------ -------- --------------------
ACCT              0 SQL130617115333860                    1                    1
HISTORY           0 SQL130617110037380                    1                    1
TELLER            0 SQL130618131822550                    1                    1

OBJECT_INDEX_L_READS OBJECT_INDEX_P_READS OBJECT_INDEX_LBP_PAGES_FOUND
-------------------- -------------------- ----------------------------
                   6                    1                            5
                   9                    1                            8
                   5                    1                            4

  3 record(s) selected.
```

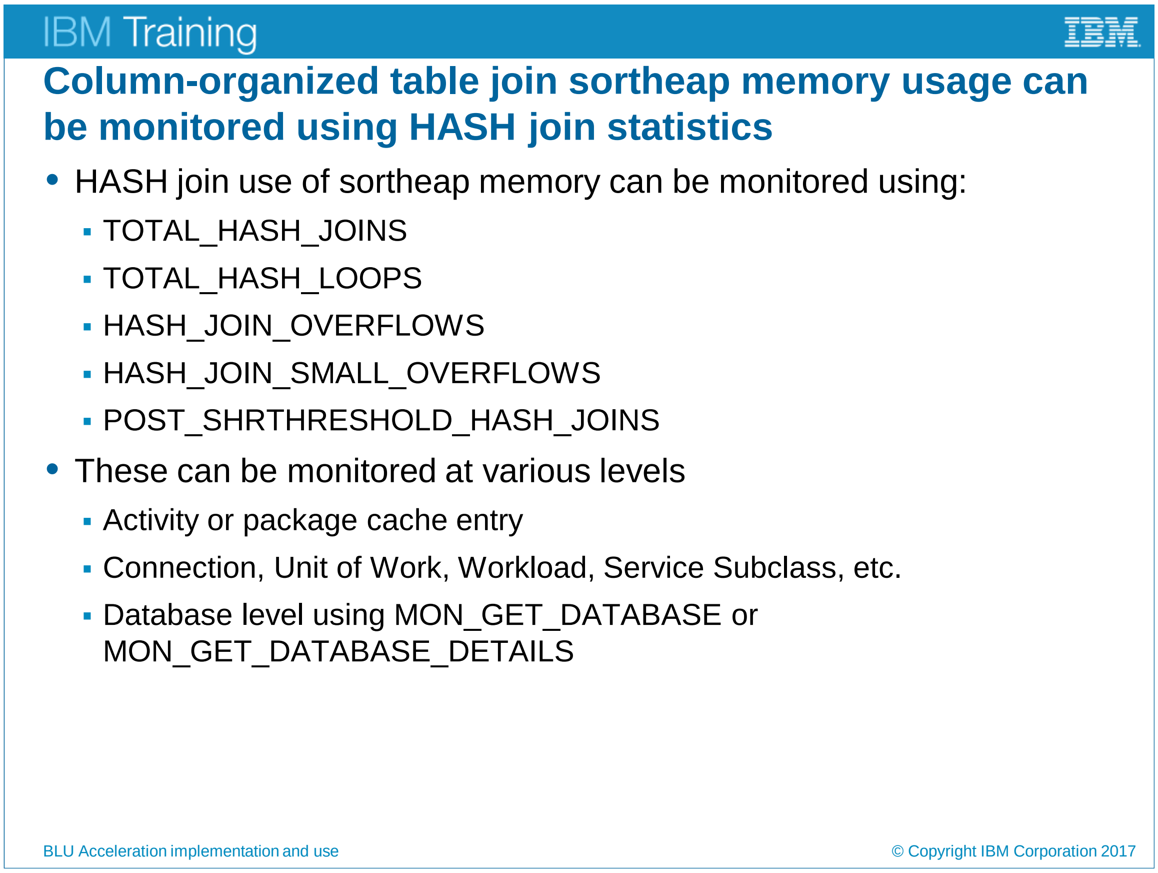
BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Monitoring Page Map Index statistics for column-organized tables using MON_GET_INDEX

The page map indexes that DB2 creates and accesses internally for column-organized tables can be monitored like any other DB2 index.

The use of page map indexes will not be included in the explain reports, but the processing of these indexes can still be monitored.

The sample query uses the statistics available through the monitor table function MON_GET_INDEX that shows buffer pool activity for index pages.



IBM Training

Column-organized table join sortheap memory usage can be monitored using HASH join statistics

- HASH join use of sortheap memory can be monitored using:
  - TOTAL_HASH_JOINS
  - TOTAL_HASH_LOOPS
  - HASH_JOIN_OVERFLOWS
  - HASH_JOIN_SMALL_OVERFLOWS
  - POST_SHRTHRESHOLD_HASH_JOINS
- These can be monitored at various levels
  - Activity or package cache entry
  - Connection, Unit of Work, Workload, Service Subclass, etc.
  - Database level using MON_GET_DATABASE or MON_GET_DATABASE_DETAILS

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

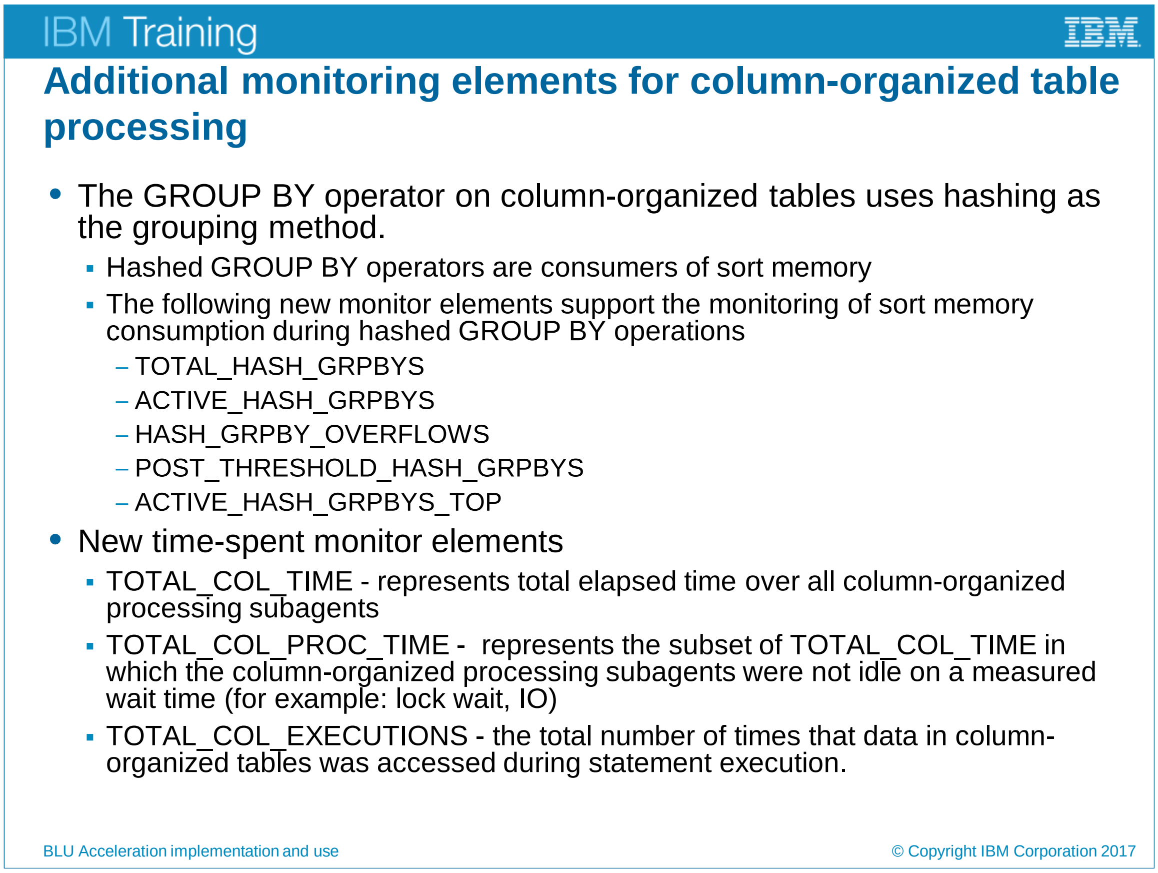
Column-organized table join sortheap memory usage can be monitored using HASH join statistics

DB2 uses a form of HASH join for joining Column-organized tables. This makes use of database sort memory during processing.

With DB2 11 monitoring hash join activity can be performed with a set of monitor elements using the monitor functions like MON_GET_ACTIVITY, MON_GET_CONNECTION and MON_GET_DATABASE.

The monitor elements related to HASH join processing are:

- TOTAL_HASH_JOINS - The total number of hash joins executed.
- TOTAL_HASH_LOOPS - The total number of times that a single partition of a hash join was larger than the available sort heap space.
- HASH_JOIN_OVERFLOWS - The number of times that hash join data exceeded the available sort heap space.
- HASH_JOIN_SMALL_OVERFLOWS - The number of times that hash join data exceeded the available sort heap space by less than 10%.
- POST_SHRTHRESHOLD_HASH_JOINS - The total number of times that a hash join heap request was limited due to concurrent use of shared sort heap space.


IBM Training

Additional monitoring elements for column-organized table processing

- The GROUP BY operator on column-organized tables uses hashing as the grouping method.
  - Hashed GROUP BY operators are consumers of sort memory
  - The following new monitor elements support the monitoring of sort memory consumption during hashed GROUP BY operations
    - TOTAL_HASH_GRPBYS
    - ACTIVE_HASH_GRPBYS
    - HASH_GRPBY_OVERFLOWS
    - POST_THRESHOLD_HASH_GRPBYS
    - ACTIVE_HASH_GRPBYS_TOP
- New time-spent monitor elements
  - TOTAL_COL_TIME - represents total elapsed time over all column-organized processing subagents
  - TOTAL_COL_PROC_TIME - represents the subset of TOTAL_COL_TIME in which the column-organized processing subagents were not idle on a measured wait time (for example: lock wait, IO)
  - TOTAL_COL_EXECUTIONS - the total number of times that data in column-organized tables was accessed during statement execution.

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Additional monitoring elements for column-organized table processing

The GROUP BY operator on column-organized tables uses hashing as the grouping method. Hashed GROUP BY operators are consumers of sort memory.

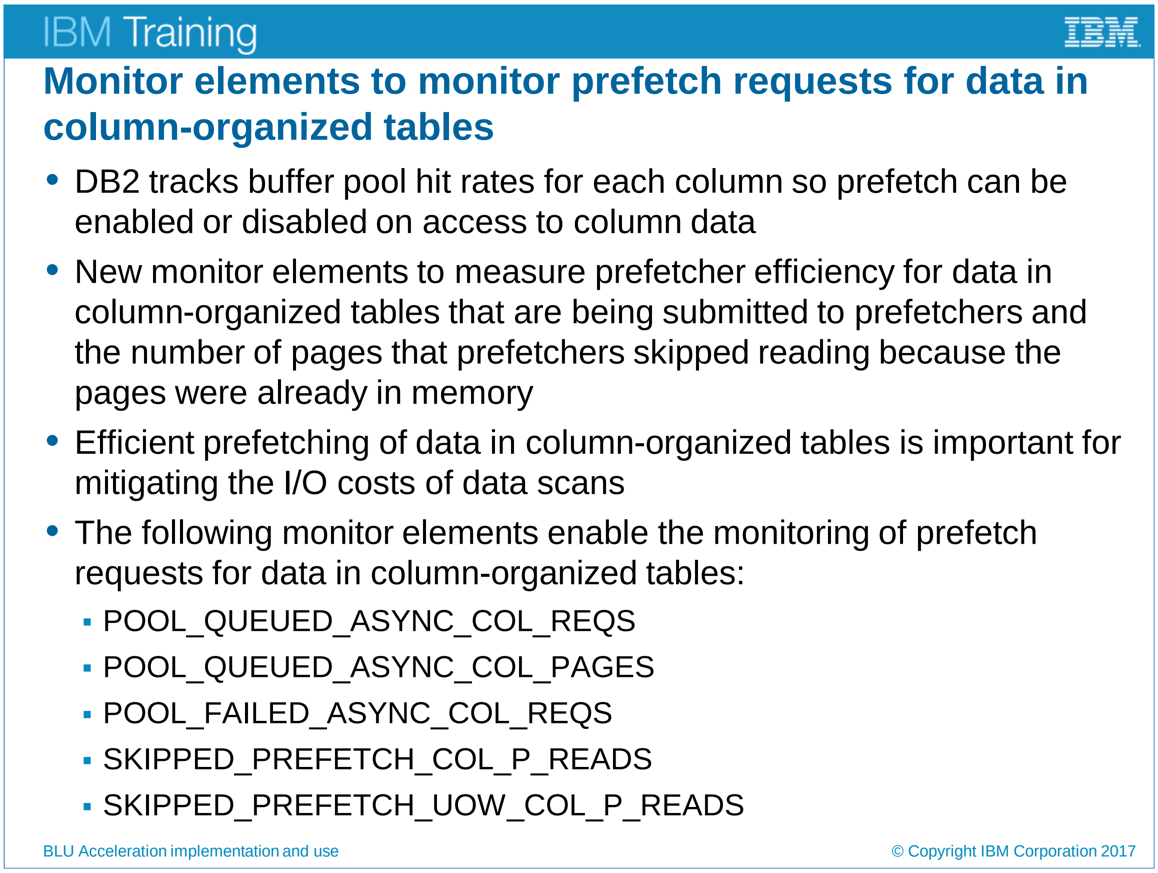
The following new monitor elements support the monitoring of sort memory consumption during hashed GROUP BY operations. These elements are similar to existing monitor elements for other sort memory consumers.

- TOTAL_HASH_GRPBYS
- ACTIVE_HASH_GRPBYS
- HASH_GRPBY_OVERFLOWS
- POST_THRESHOLD_HASH_GRPBYS

Time-spent monitor elements provide information about how the DB2 database manager is spending time processing column-organized tables. The time-spent elements are broadly categorized into wait times and processing times.

The following monitor elements are added to the time-spent hierarchy:

- The three TOTAL_* metrics count the total time that is spent in column-organized data processing across all column-organized processing subagents.
- **TOTAL_COL_TIME** represents total elapsed time over all column-organized processing subagents.
- **TOTAL_COL_PROC_TIME** represents the subset of TOTAL_COL_TIME in which the column-organized processing subagents were not idle on a measured wait time (for example: lock wait, IO).
- **TOTAL_COL_EXECUTIONS** represents the total number of times that data in column-organized tables was accessed during statement execution.



IBM Training

Monitor elements to monitor prefetch requests for data in column-organized tables

- DB2 tracks buffer pool hit rates for each column so prefetch can be enabled or disabled on access to column data
- New monitor elements to measure prefetcher efficiency for data in column-organized tables that are being submitted to prefetchers and the number of pages that prefetchers skipped reading because the pages were already in memory
- Efficient prefetching of data in column-organized tables is important for mitigating the I/O costs of data scans
- The following monitor elements enable the monitoring of prefetch requests for data in column-organized tables:
  - POOL_QUEUED_ASYNC_COL_REQS
  - POOL_QUEUED_ASYNC_COL_PAGES
  - POOL_FAILED_ASYNC_COL_REQS
  - SKIPPED_PREFETCH_COL_P_READS
  - SKIPPED_PREFETCH_UOW_COL_P_READS

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Monitor elements to monitor prefetch requests for data in column-organized tables

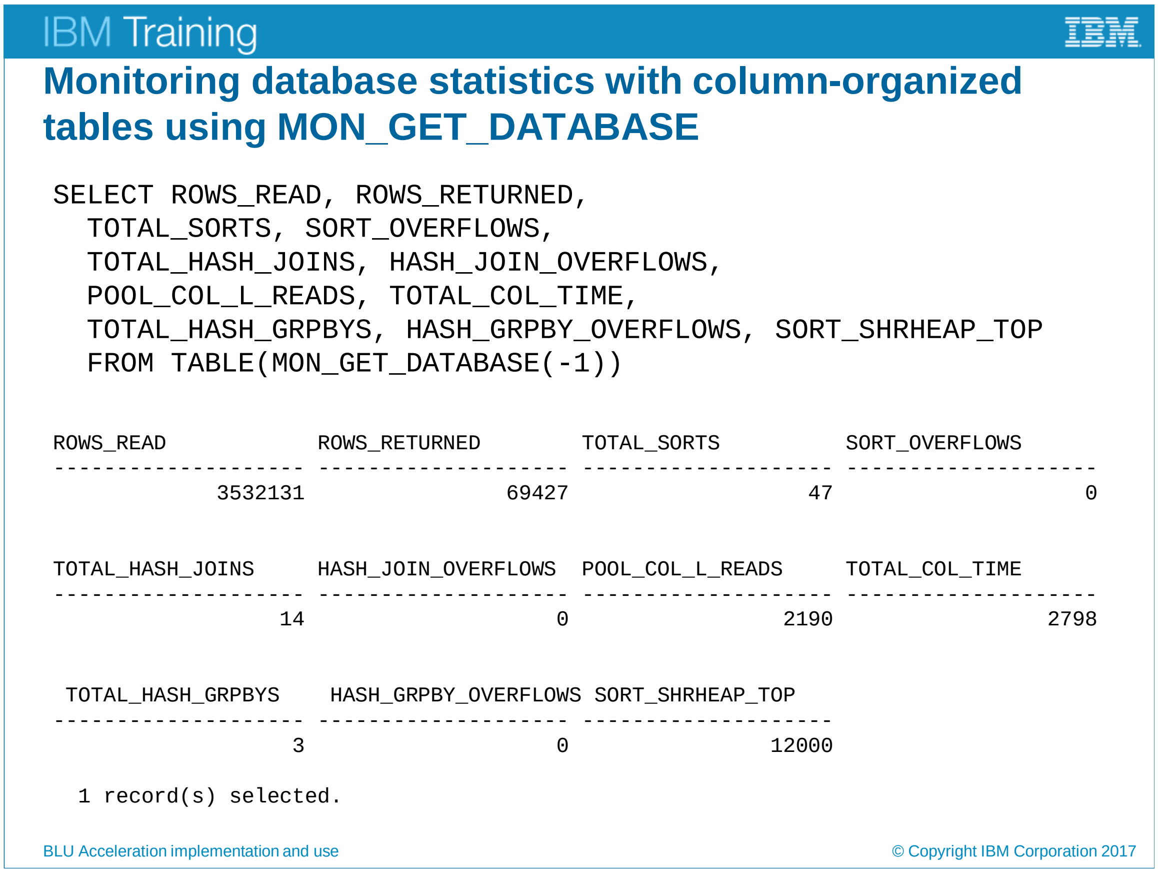
The prefetch logic for queries that access column-organized tables is used to asynchronously fetch only those pages that each thread will read for each column that is accessed during query execution. If the pages for a particular column are consistently available in the buffer pool, prefetching for that column is disabled until the pages are being read synchronously, at which time prefetching for that column is enabled again.

Although the number of pages that a thread can prefetch simultaneously is limited by the prefetch size of the table space that is being accessed, several threads can also prefetch pages simultaneously.

The monitor elements to measure prefetcher efficiency can help you to track the volume of requests for data in column-organized tables that are being submitted to prefetchers, and the number of pages that prefetchers skipped reading because the pages were already in memory. Efficient prefetching of data in column-organized tables is important for mitigating the I/O costs of data scans.

The following monitor elements enable the monitoring of prefetch requests for data in column-organized tables:

- POOL_QUEUED_ASYNC_COL_REQS
- POOL_QUEUED_ASYNC_COL_PAGES
- POOL_FAILED_ASYNC_COL_REQS
- SKIPPED_PREFETCH_COL_P_READS
- SKIPPED_PREFETCH_UOW_COL_P_READS



IBM Training

Monitoring database statistics with column-organized tables using MON_GET_DATABASE

```
SELECT ROWS_READ, ROWS_RETURNED,
  TOTAL_SORTS, SORT_OVERFLOWS,
  TOTAL_HASH_JOINS, HASH_JOIN_OVERFLOWS,
  POOL_COL_L_READS, TOTAL_COL_TIME,
  TOTAL_HASH_GRPBYS, HASH_GRPBY_OVERFLOWS, SORT_SHRHEAP_TOP
  FROM TABLE(MON_GET_DATABASE(-1))
```

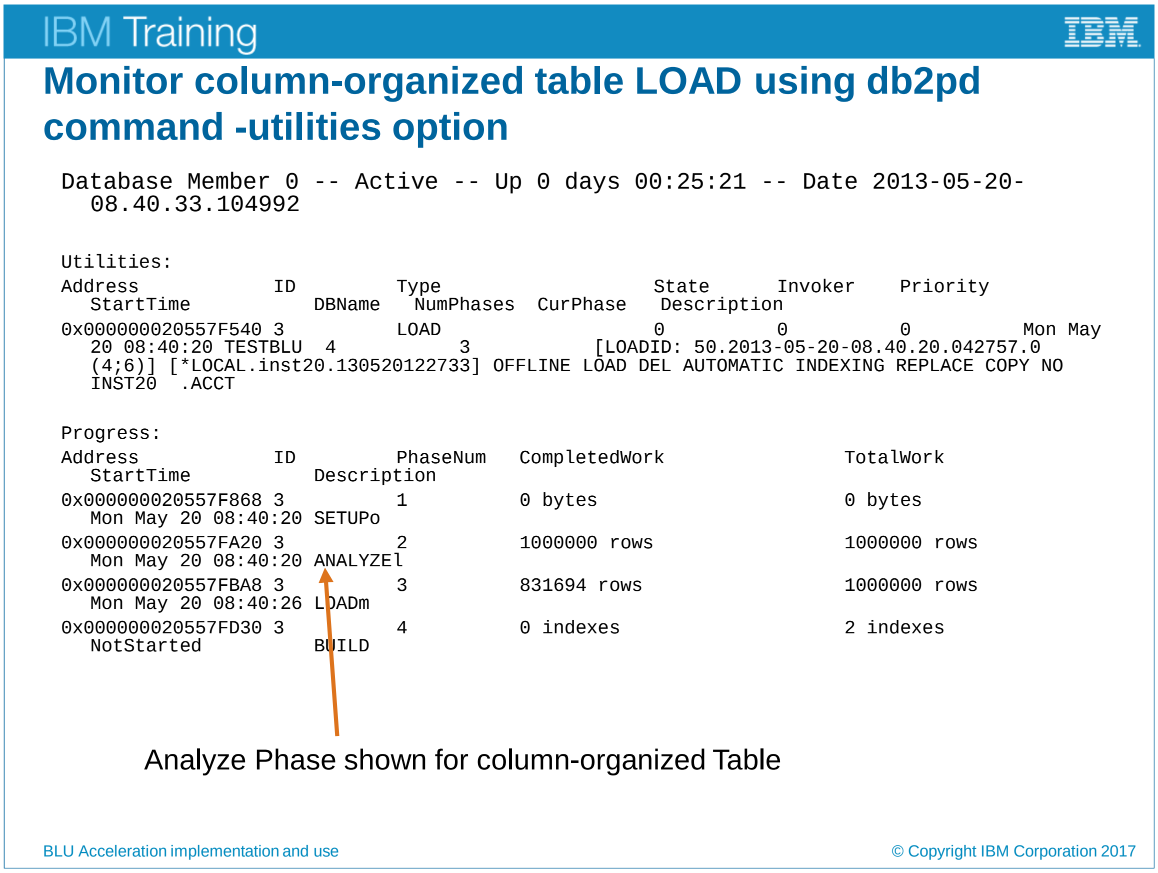
```
ROWS_READ            ROWS_RETURNED        TOTAL_SORTS          SORT_OVERFLOWS
-------------------- -------------------- -------------------- --------------------
             3532131                69427                   47                    0

TOTAL_HASH_JOINS     HASH_JOIN_OVERFLOWS  POOL_COL_L_READS     TOTAL_COL_TIME
-------------------- -------------------- -------------------- --------------------
                  14                    0                 2190                 2798

 TOTAL_HASH_GRPBYS    HASH_GRPBY_OVERFLOWS SORT_SHRHEAP_TOP
-------------------- -------------------- --------------------
                   3                    0                12000

  1 record(s) selected.
```

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Monitoring database statistics with column-organized tables using MON_GET_DATABASE

The sample query uses the MON_GET_DATABASE table function to retrieve some of the monitoring elements that would indicate some key performance measures for column-organized processing, like hash joins, hash based group by processing.

These could be used to monitor efficient configuration of database shared sort memory, which cannot be managed by the self-tuning memory manager with column-organized table support.



IBM Training

Monitor column-organized table LOAD using db2pd command -utilities option

```
Database Member 0 -- Active -- Up 0 days 00:25:21 -- Date 2013-05-20-
   08.40.33.104992


Utilities:
Address            ID          Type                   State      Invoker    Priority
   StartTime          DBName   NumPhases  CurPhase   Description
0x000000020557F540 3           LOAD                   0          0          0          Mon May
   20 08:40:20 TESTBLU  4          3           [LOADID: 50.2013-05-20-08.40.20.042757.0
   (4;6)] [*LOCAL.inst20.130520122733] OFFLINE LOAD DEL AUTOMATIC INDEXING REPLACE COPY NO
   INST20   .ACCT


Progress:
Address            ID          PhaseNum   CompletedWork                TotalWork
   StartTime          Description
0x000000020557F868 3           1          0 bytes                      0 bytes
   Mon May 20 08:40:20 SETUPo
0x000000020557FA20 3           2          1000000 rows                 1000000 rows
   Mon May 20 08:40:20 ANALYZEl
0x000000020557FBA8 3           3          831694 rows                  1000000 rows
   Mon May 20 08:40:26 LOADm
0x000000020557FD30 3           4          0 indexes                    2 indexes
   NotStarted         BUILD
```

Analyze Phase shown for column-organized Table

BLU Acceleration implementation and use

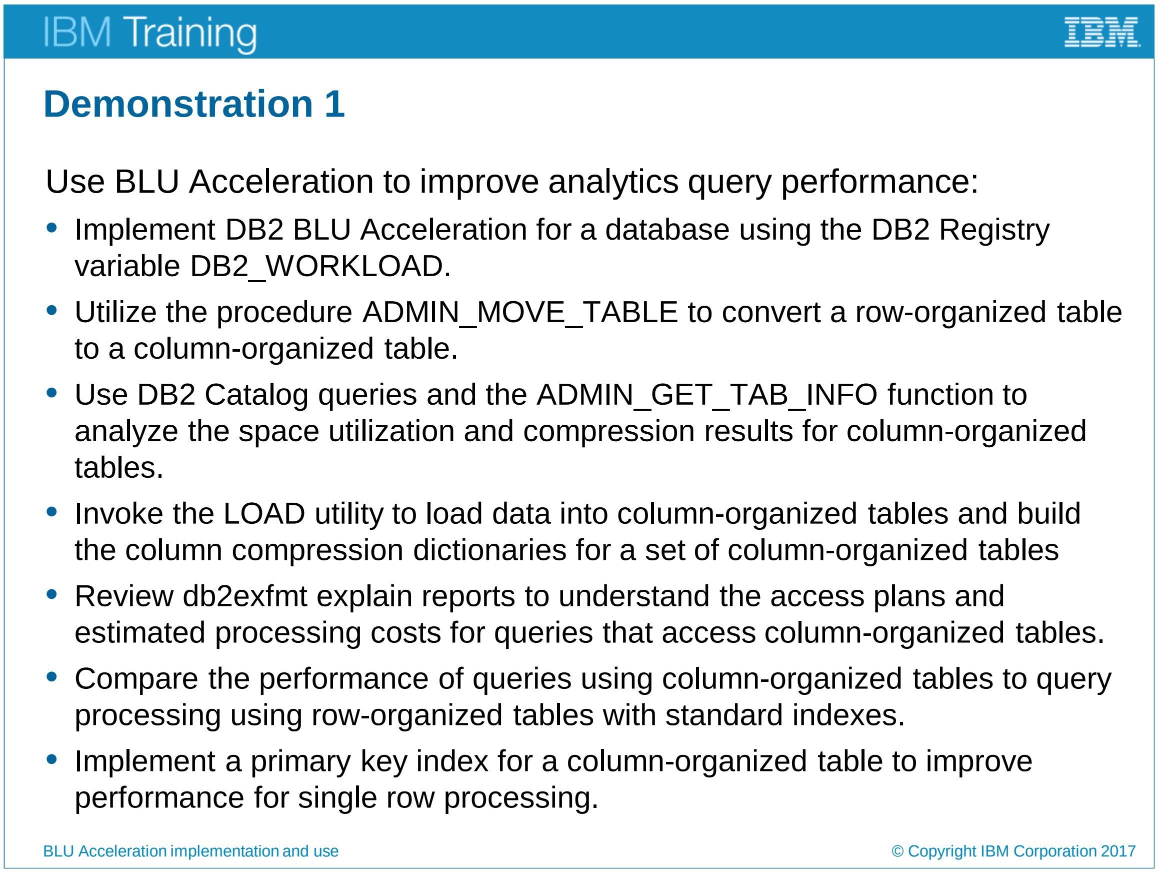
© Copyright IBM Corporation 2017

Monitor column-organized table LOAD using db2pd command -utilities option

The processing performed by the LOAD utility for column-organized tables is a key component for column-organized table support. The column dictionaries are created by a LOAD utility using a new phase of processing, ANALYZE.

The sample db2pd command report for the -utilities option can be used to monitor LOAD utility processing for column-organized tables. The example report output shows that the LOAD utility has completed the ANALYZE phase and is currently performing the LOAD phase of processing. Compared to load processing for row-organized tables, the BUILD phase should be less time-consuming, since the page map indexes used for column-organized tables are built on a page basis not a row basis.

The memory requirements for loading column-organized tables may require some changes to the number of load utilities that are run concurrently.



IBM Training

Demonstration 1

Use BLU Acceleration to improve analytics query performance:

- Implement DB2 BLU Acceleration for a database using the DB2 Registry variable DB2_WORKLOAD.
- Utilize the procedure ADMIN_MOVE_TABLE to convert a row-organized table to a column-organized table.
- Use DB2 Catalog queries and the ADMIN_GET_TAB_INFO function to analyze the space utilization and compression results for column-organized tables.
- Invoke the LOAD utility to load data into column-organized tables and build the column compression dictionaries for a set of column-organized tables
- Review db2exfmt explain reports to understand the access plans and estimated processing costs for queries that access column-organized tables.
- Compare the performance of queries using column-organized tables to query processing using row-organized tables with standard indexes.
- Implement a primary key index for a column-organized table to improve performance for single row processing.

BLU Acceleration implementation and use © Copyright IBM Corporation 2017

Demonstration 1: Use BLU Acceleration to improve analytics query performance

Demonstration 1:
Use BLU Acceleration to improve analytics query performance

Purpose:
This demonstration will show you how to implement DB2 with BLU Acceleration for a new database. We will compare the disk space requirements for column-organized tables to standard row-organized tables, with and without compression. We will run a set of sample queries to compare the access plans and performance characteristics of query processing using DB2 with BLU Acceleration with the processing techniques used for row-organized tables with indexes.

Task 1. Setup a new DB2 database for testing DB2 BLU acceleration for query processing.

Important Note: This demonstration will create a new database DB2BLU for implementation of the BLU Acceleration feature. The DB2 instance **inst450** has already been created and configured with the DB2 Registry variable DB2_WORKLOAD set to ANALYTICS. This will impact the initial configuration of the new database. You need to logon to the Linux system with the DB2 instance owner id: inst450.

These advanced labs are performed using command line Linux and DB2 commands. You will use the Linux Terminal session to enter DB2 commands

1. From your Linux workstation, login with the userid of **inst450** and the password provided by the instructor (default ***ibm2blue***). If the desktop shows only the screensaver wallpaper, press the **Enter** key to bring up the userid list. Throughout the lab, do the same if the screensaver wallpaper appears, to return to the desktop.
2. Right-click the empty Linux desktop, and then select **Open in Terminal**.

   The DB2 instance for this demonstration (and the DB2 instance for the later MPP demonstration) need to be configured with a valid tcpip address assigned to the system host name **ibmclass** in the */etc/hosts* file.

   You will login to the system root user using the Linux su command and check the tcpip configuration. The default password for root is **dalvm3**.
3. Enter the following commands in the Linux terminal session:
   - **su - root (default password is dalvm3)**
   - **ifconfig**

The ifconfig output will be similar to the following:

```
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.164.133  netmask 255.255.255.0  broadcast 192.168.164.255
        inet6 fe80::20c:29ff:fe24:5503  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:24:55:03  txqueuelen 1000  (Ethernet)
        RX packets 715  bytes 51847 (50.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 231  bytes 23629 (23.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 0  (Local Loopback)
        RX packets 8  bytes 680 (680.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 680 (680.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

virbr0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.122.1  netmask 255.255.255.0  broadcast 192.168.122.255
        ether 52:54:00:70:b5:f7  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Note the inet address for the Ethernet adapter on your system and use the system **vi** editor to update the file **/etc/hosts** and assign this tcpip address to the hostname ibmclass. You may prefer to use another editor, like **gedit**.

The hosts file will have an entry for ibmclass similar to the following:

```
192.168.164.133 ibmclass ibmclass.class.com
```

4. Enter the following commands in the Linux terminal session:

- **vi /etc/hosts**
- *(change the existing entry to the inet address found above and save the updated file)*
- **exit (at the linux command prompt)**

You have returned to the inst450 command prompt

5. Enter the following commands in the Linux terminal session:
   - **cd $HOME/db2blu**
   - **db2set -all**

   The output from this command will be similar to the following:
   ```
   [i] DB2_WORKLOAD=analytics
   [i] DB2_USE_ALTERNATE_PAGE_CLEANING=ON [DB2_WORKLOAD]
   [i] DB2_ANTIJOIN=EXTEND [DB2_WORKLOAD]
   [i] DB2COMM=TCPIP
   [g] DB2SYSTEM=ibmclass
   [g] DB2INSTDEF=db2inst1
   ```
   - **db2start**
   - **db2 terminate**
   - **db2 create database db2blu on /database**

   With the DB2 Registry variable DB2_WORKLOAD set to the value **ANALYTICS** (or **analytics**) before the database is created, a newly created database is automatically configured for BLU Acceleration. For example, the database page size will be set to 32K rather than the 4K default and the default buffer pool will have a 32K page size.

   Use the command file *checkcfg.sql*, which uses the view SYSIBMADM.DBCFG to list some of the database configuration options tailored to the analytics workload used for DB2 with BLU Acceleration.

6. In the terminal session, enter the following commands:
   - **db2 connect to db2blu**
   - **db2 -tvf checkcfg.sql**

   The output from this command will be similar to the following:
   ```
   select name as CFG_option, varchar(value,30)  as configured_value  from
   sysibmadm.dbcfg
   where name in ('sheapthres_shr','sortheap','util_heap_sz','dft_table_org',
   'self_tuning_mem','num_ioservers')
   order by name

   CFG_OPTION                       CONFIGURED_VALUE
   -------------------------------- ------------------------------
   dft_table_org                    COLUMN
   num_ioservers                    12
   self_tuning_mem                  ON (Active)
   sheapthres_shr                   153546
   sortheap                         32768
   util_heap_sz                     46506

     6 record(s) selected.
   ```

Notice that the option DFT_TABLE_ORG is set to COLUMN, so tables will be created as column-organized by default.

The database is configured for self-tuning memory management, but has the sort memory options SORTHEAP and SHEAPTHRES_SHR set manually since these cannot be self-tuned with BLU acceleration.

We are going to create two sets of tables, one set defined with a schema of COLORG, and one set of row-organized tables using the schema ROWORG. First we will create table spaces to store the tables and indexes. The file **tspace.ddl** creates four new table spaces.

7. In the terminal session, enter the following command:

   - **db2 -tvf tspace.ddl**

   The output from this command will be similar to the following:

```
CREATE TABLESPACE TSCOLD
DB20000I  The SQL command completed successfully.

CREATE TABLESPACE TSCOLI
DB20000I  The SQL command completed successfully.

CREATE TABLESPACE TSROWD
DB20000I  The SQL command completed successfully.

CREATE TABLESPACE TSROWI
DB20000I  The SQL command completed successfully.
```

   We will initially create the first table COLORG.HISTORY as a row-organized table with standard indexes and then use ADMIN_MOVE_TABLE to convert the table to a column-organized table. Use the file *create_colhist.ddl* to create the table COLORG.HISTORY.

8. In the terminal session, enter the following command:
   - **db2 -tvf create_colhist.ddl**

   The output from this command will be similar to the following:

```
set current schema = 'COLORG'
DB20000I  The SQL command completed successfully.

CREATE TABLE HISTORY (ACCT_ID          INTEGER         NOT NULL, TELLER_ID
SMALLINT        NOT NULL, BRANCH_ID        SMALLINT        NOT NULL, BALANCE
DECIMAL(15,2)   NOT NULL, DELTA            DECIMAL(9,2)    NOT NULL, PID
INTEGER         NOT NULL, TSTMP            TIMESTAMP       NOT NULL WITH DEFAULT,
ACCTNAME         CHAR(20)        NOT NULL, TEMP             CHAR(6)         NOT
NULL) organize by row IN TSROWD INDEX IN TSROWI
DB20000I  The SQL command completed successfully.

CREATE INDEX HISTIX1 ON HISTORY (BRANCH_ID ASC) ALLOW REVERSE SCANS
DB20000I  The SQL command completed successfully.

CREATE INDEX HISTIX2 ON HISTORY (TELLER_ID ASC) ALLOW REVERSE SCANS
DB20000I  The SQL command completed successfully.
```

   The *organize by row* clause is necessary in the DB2BLU database since the default table organization was set to COLUMN.

   Next we will use the LOAD utility to load a set of test data into the table.

   The file *loadhistory.sql* contains the statements to load the table COLORG.HISTORY.

9. In the terminal session, enter the following command:
   - **db2 -tvf loadhistory.sql**

   The output from this command will be similar to the following:

```
load from histdata.del of del messages loadhist1.msg replace into colorg.history

Number of rows read         = 490864
Number of rows skipped      = 0
Number of rows loaded       = 490864
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 490864
```

   We can use ADMIN_MOVE_TABLE to convert the row-organized table to a column-organized table. The two indexes we created on the table COLORG.HISTORY are not unique indexes, so they will not be created when the table is redefined. The file *convert_history.sql* contains the CALL statement for ADMIN_MOVE_TABLE to redefine the table as column-organized and changes the table space assignments. We will not keep a copy of the original table.

10. In the terminal session, enter the following command:

    - **db2 -tvf convert_history.sql | more**

    The output from this command will be roughly similar to the following (keys may be output in a different order):

```
call SYSPROC.ADMIN_MOVE_TABLE ( 'COLORG', 'HISTORY', 'TSCOLD','TSCOLI','TSCOLD',
'ORGANIZE BY COLUMN',NULL,NULL,NULL, 'COPY_USE_LOAD,FORCE','MOVE')


  Result set 1
  --------------

  KEY                              VALUE
  -------------------------------- -----------------------------------------------
--------------------------------------------------------------------------------
  AUTHID                           INST450
  CLEANUP_END                      2017-05-03-15.52.37.770636
  COPY_OPTS                        LOAD,NON_CLUSTER
  COPY_START                       2017-05-03-15.52.27.538097
  COPY_TOTAL_ROWS                  490864
  DICTIONARY_CREATION_TOTAL_TIME   5
  INDEX_CREATION_TOTAL_TIME        0
  INIT_END                         2017-05-03-15.52.27.437872
  ORIGINAL_TBLSIZE                 43776
  REPLAY_END                       2017-05-03-15.52.37.603545
  REPLAY_TOTAL_TIME                1
  STATUS                           COMPLETE
  SWAP_START                       2017-05-03-15.52.37.604132
  UTILITY_INVOCATION_ID
0100000043000000080000000000000000201705031552274410280000000
  VERSION                          11.01.0101
  CLEANUP_START                    2017-05-03-15.52.37.699123
  COPY_END                         2017-05-03-15.52.36.163038
  INDEXNAME                        HISTIX2
  INDEXSCHEMA                      COLORG
  INIT_START                       2017-05-03-15.52.26.593755
  REPLAY_START                     2017-05-03-15.52.36.163951
  REPLAY_TOTAL_ROWS                0
  SWAP_END                         2017-05-03-15.52.37.645397
  SWAP_RETRIES                     0

  24 record(s) selected.

  Return Status = 0
```

The COPY_USE_LOAD option is important when converting to a column-organized table since the LOAD ANALYZE phase builds the column dictionaries using a full scan of the data. The new table COLORG.HISTORY has catalog statistics already collected, so a RUNSTATS is not necessary.

While ADMIN_MOVE_TABLE can be used to simplify converting a row-organized table to a column-organized table, it cannot be used to convert a column-organized table to a row-organized table.

You will create the other three column-organized tables, ACCT, BRANCH and TELLER using the COLORG schema and load them using the LOAD utility. Use the DB2 command file *tables.ddl* to create the tables.

11. In the terminal session, enter the following command:
    - **db2 -tvf tables.ddl**

    The output from this command will be similar to the following:

```
set current schema = 'COLORG'
DB20000I  The SQL command completed successfully.

CREATE TABLE ACCT (ACCT_ID          INT             NOT NULL, NAME
CHAR(20)        NOT NULL, ACCT_GRP          SMALLINT        NOT NULL, BALANCE
DECIMAL(15,2)   NOT NULL, ADDRESS           CHAR(30)        NOT NULL, TEMP
CHAR(40)        NOT NULL) IN TSCOLD INDEX IN TSCOLI
DB20000I  The SQL command completed successfully.

CREATE TABLE BRANCH (BRANCH_ID        SMALLINT        NOT NULL, BRANCH_NAME
CHAR(20)        NOT NULL, BALANCE           DECIMAL(15,2)   NOT NULL, AREA_CODE
CHAR(4)         NOT NULL, ADDRESS           CHAR(30)        NOT NULL, TEMP
CHAR(40)        NOT NULL) IN TSCOLD INDEX IN TSCOLI
DB20000I  The SQL command completed successfully.

CREATE TABLE TELLER (TELLER_ID        SMALLINT        NOT NULL, TELLER_NAME
CHAR(20)        NOT NULL, BRANCH_ID         SMALLINT        NOT NULL, BALANCE
DECIMAL(15,2)   NOT NULL, TELLER_CODE       CHAR(2)         NOT NULL, ADDRESS
CHAR(30)        NOT NULL, TEMP              CHAR(40)        NOT NULL) IN TSCOLD
INDEX IN TSCOLI
DB20000I  The SQL command completed successfully.
```

    Next we will use the LOAD utility to load a set of test data into the table. The file *loadacct.sql* contains the statements to load the table COLORG.ACCT.

12. In the terminal session, enter the following command:
    - **db2 -tvf loadacct.sql**

    The output from this command will be similar to the following:

```
load from acct.del of del messages loadacctc.msg replace into colorg.acct

Number of rows read         = 1000000
Number of rows skipped      = 0
Number of rows loaded       = 1000000
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 1000000
```

    Review the messages generated by the LOAD utility processing for the table COLORG.ACCT.

13. In the terminal session, enter the following command:
    - **cat loadacctc.msg**

    The output from this command will be similar to the following:

```
SQL3501W  The table space(s) in which the table resides will not be placed in
backup pending state since forward recovery is disabled for the database.

SQL3109N  The utility is beginning to load data from file
"/home/inst450/db2blu/acct.del".

SQL3500W  The utility is beginning the "ANALYZE" phase at time "05/03/2017
16:04:50.270420".

SQL3519W  Begin Load Consistency Point. Input record count = "0".

SQL3520W  Load Consistency Point was successful.

SQL3515W  The utility has finished the "ANALYZE" phase at time "05/03/2017
16:04:55.539125".

SQL3500W  The utility is beginning the "LOAD" phase at time "05/03/2017
16:04:55.541193".

SQL3110N  The utility has completed processing.  "1000000" rows were read from
the input file.

SQL3519W  Begin Load Consistency Point. Input record count = "1000000".

SQL3520W  Load Consistency Point was successful.

SQL3515W  The utility has finished the "LOAD" phase at time "05/03/2017
```

```
16:05:00.136401".

SQL3500W  The utility is beginning the "BUILD" phase at time "05/03/2017
16:05:00.138690".

SQL3213I  The indexing mode is "REBUILD".

SQL3515W  The utility has finished the "BUILD" phase at time "05/03/2017
16:05:00.223045".


Number of rows read         = 1000000
Number of rows skipped      = 0
Number of rows loaded       = 1000000
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 1000000
```

Note that LOAD processing for the column-organized table includes the ANALYZE phase, needed to build the column compression dictionaries.

The tables BRANCH and TELLER will be loaded using the command file *loadothers.sql*.

14. In the terminal session, enter the following command:
    - **db2 -tvf loadothers.sql**

    The output from this command will be similar to the following:

```
load from branch.del of del messages loadbranch.msg replace into COLORG.branch

Number of rows read         = 100
Number of rows skipped      = 0
Number of rows loaded       = 100
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 100


load from teller.del of del messages loadteller.msg replace into COLORG.teller

Number of rows read         = 1000
Number of rows skipped      = 0
Number of rows loaded       = 1000
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 1000
```

Next you will create four row-organized tables and load them with the same test data used for the column-organized tables. The tables will use the schema ROWORG. The ACCT table will be created using adaptive compression. The other three row-organized tables, HISTORY, BRANCH and TELLER will not be defined as compressed. Use the DB2 command file *tablesrow.ddl* to create the tables.

15. In the terminal session, enter the following command:

    - **db2 -tvf tablesrow.ddl**

    The output from this command will be similar to the following:

```
set current schema = 'ROWORG'
DB20000I  The SQL command completed successfully.

CREATE TABLE ACCT (ACCT_ID          INT             NOT NULL, NAME
CHAR(20)        NOT NULL, ACCT_GRP         SMALLINT        NOT NULL, BALANCE
DECIMAL(15,2)   NOT NULL, ADDRESS          CHAR(30)        NOT NULL, TEMP
CHAR(40)        NOT NULL) COMPRESS YES ADAPTIVE organize by row IN TSROWD INDEX IN
TSROWI
DB20000I  The SQL command completed successfully.

alter table acct add primary key (acct_id)
DB20000I  The SQL command completed successfully.

CREATE TABLE HISTORY (ACCT_ID          INTEGER         NOT NULL, TELLER_ID
SMALLINT        NOT NULL, BRANCH_ID        SMALLINT        NOT NULL, BALANCE
DECIMAL(15,2)   NOT NULL, DELTA            DECIMAL(9,2)    NOT NULL, PID
INTEGER         NOT NULL, TSTMP            TIMESTAMP       NOT NULL WITH DEFAULT,
ACCTNAME         CHAR(20)        NOT NULL, TEMP             CHAR(6)         NOT
NULL) organize by row IN TSROWD INDEX IN TSROWI
DB20000I  The SQL command completed successfully.

CREATE INDEX HISTIX1 ON HISTORY (BRANCH_ID ASC) ALLOW REVERSE SCANS
DB20000I  The SQL command completed successfully.

CREATE INDEX HISTIX2 ON HISTORY (TELLER_ID ASC) ALLOW REVERSE SCANS
DB20000I  The SQL command completed successfully.

CREATE TABLE BRANCH (BRANCH_ID        SMALLINT        NOT NULL, BRANCH_NAME
CHAR(20)        NOT NULL, BALANCE          DECIMAL(15,2)   NOT NULL, AREA_CODE
CHAR(4)         NOT NULL, ADDRESS          CHAR(30)        NOT NULL, TEMP
CHAR(40)        NOT NULL) organize by row IN TSROWD
DB20000I  The SQL command completed successfully.

alter table branch add primary key (branch_id)
DB20000I  The SQL command completed successfully.
```

```
CREATE TABLE TELLER (TELLER_ID       SMALLINT        NOT NULL, TELLER_NAME
CHAR(20)        NOT NULL, BRANCH_ID        SMALLINT        NOT NULL, BALANCE
DECIMAL(15,2)   NOT NULL, TELLER_CODE      CHAR(2)         NOT NULL, ADDRESS
CHAR(30)        NOT NULL, TEMP             CHAR(40)        NOT NULL) organize by
row IN TSROWD
DB20000I  The SQL command completed successfully.

alter table teller add primary key (teller_id)
DB20000I  The SQL command completed successfully.
```

Several indexes were defined on these tables using columns that will benefit the sample queries used for testing and also support joining the tables together. Use the file *loadrowtabs.ddl* to load the four tables and collect statistics using RUNSTATS.

16. In the terminal session, enter the following command:
    - **db2 -tvf loadrowtabs.ddl**

    The output from this command will be similar to the following:

```
load from acct.del of del messages loadracct.msg replace into roworg.acct

Number of rows read         = 1000000
Number of rows skipped      = 0
Number of rows loaded       = 1000000
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 1000000


runstats on table roworg.acct and indexes all
DB20000I  The RUNSTATS command completed successfully.

load from histdata.del of del messages loadrhist.msg replace into roworg.history

Number of rows read         = 490864
Number of rows skipped      = 0
Number of rows loaded       = 490864
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 490864


runstats on table roworg.history and indexes all
DB20000I  The RUNSTATS command completed successfully.

load from branch.del of del messages loadrbranch.msg replace into ROWORG.branch
```

```
Number of rows read         = 100
Number of rows skipped      = 0
Number of rows loaded       = 100
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 100


runstats on table roworg.branch and indexes all
DB20000I  The RUNSTATS command completed successfully.

load from teller.del of del messages loadrteller.msg replace into ROWORG.teller

Number of rows read         = 1000
Number of rows skipped      = 0
Number of rows loaded       = 1000
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 1000


runstats on table roworg.teller and indexes all
DB20000I  The RUNSTATS command completed successfully.
```

The RUNSTATS utility was run to collect table and index statistics for each table.

An important aspect of DB2 BLU Acceleration is the extreme compression for larger tables. We will use several queries to compare the storage requirements for the two sets of tables. All column-organized tables are compressed using the column compression dictionaries. We defined the table ROWORG.ACCT to utilize Adaptive compression. Use the DB2 command file *qsyscat.sql* to query the statistics in the catalog table SYSCAT.TABLES.

17. In the terminal session, enter the following command:
    - **db2 -tvf qsyscat.sql**

    The output from this command will be similar to the following:

```
select varchar(tabschema,12) as schema,
 varchar(tabname,12) as table, card, tableorg,
 npages, fpages, mpages, pctpagessaved from syscat.tables
 where tabschema in ('COLORG','ROWORG')
 order by tabname

SCHEMA       TABLE        CARD                 TABLEORG NPAGES
FPAGES               MPAGES               PCTPAGESSAVED
------------ ------------ -------------------- -------- -------------------- -----
--------------- -------------------- -------------
COLORG       ACCT                      1000000 C                          140
141                   1            96
ROWORG       ACCT                      1000000 R                          581
584                   0            83
COLORG       BRANCH                        100 C                            8
9                   1            0
ROWORG       BRANCH                        100 R                            1
2                   0            0
COLORG       HISTORY                    490864 C                          188
198                  10            81
ROWORG       HISTORY                    490864 R                         1068
1069                   0            0
COLORG       TELLER                       1000 C                            9
10                   1            0
ROWORG       TELLER                       1000 R                            4
5                   0            0

  8 record(s) selected.
```

    The column-organized versions of the larger tables ACCT and HISTORY are significantly smaller than the row-organized versions. The table ROWORG.ACCT used adaptive compression and achieved 83 percent compression, while the column-organized table, COLORG.ACCT shows 96 percent compression.

    The two small column-organized tables, BRANCH and TELLER are larger than the row-organized versions. The allocation of unique extents for each column in a column-organized table can cause a small table to allocate a number of unused pages.

    The DB2 table function ADMIN_GET_TAB_INFO can be used in a query to access table statistics that are not included in catalog tables. Use the DB2 command file *tabinfocol.sql* to query these table statistics.

18. In the terminal session, enter the following command:

    - **db2 -tvf tabinfocol.sql**

    The output from this command will be similar to the following:

```
select SUBSTR(TABSCHEMA,1,10) AS SCHEMA ,
 SUBSTR(TABNAME,1,12) AS TABLE , DATA_OBJECT_P_SIZE, INDEX_OBJECT_P_SIZE ,
COL_OBJECT_P_SIZE, COL_OBJECT_L_SIZE
 from table ( admin_get_tab_info (NULL,NULL  ) ) AS TABINFO
 where tabschema in ('COLORG','ROWORG')
 order by TABNAME

SCHEMA     TABLE        DATA_OBJECT_P_SIZE   INDEX_OBJECT_P_SIZE
COL_OBJECT_P_SIZE    COL_OBJECT_L_SIZE
---------- ----------- -------------------- -------------------- ----------------
---- --------------------
COLORG     ACCT                         256                  256
5376                 5376
ROWORG     ACCT                       18816                16768
0                    0
COLORG     BRANCH                       256                  256
1152                 1152
ROWORG     BRANCH                       256                  256
0                    0
COLORG     HISTORY                      512                  256
6784                 6784
ROWORG     HISTORY                    34432                 9344
0                    0
COLORG     TELLER                       256                  256
1280                 1280
ROWORG     TELLER                       384                  256
0                    0

  8 record(s) selected.
```

    The column-organized tables show most of the disk space is allocated in the column-organized object for the tables, the columns COL_OBJECT_P_SIZE and COL_OBJECT_L_SIZE. The column dictionaries and other metadata are allocated in the data object, shown as DATA_OBJECT_P_SIZE. The page map index for a column-organized table is shown as a small amount of space in the index object for each table, shown as INDEX_OBJECT_P_SIZE.

    The amounts shown are kilobytes.

    The larger row-organized tables ACCT and HISTORY show larger allocations for the data objects and index objects and no allocation for the column-organized object.

Task 2. Compare query processing for row-organized tables to tables using BLU Acceleration.

Next, you will compare various performance characteristics of SQL queries using DB2 with BLU Acceleration with the same queries based on the row-organized tables that have standard indexes. We will look at the access plans using the **db2exfmt** explain tool to review the pre-execution estimated processing costs.

The first SQL query that we will analyze accesses a single table and produces a summarized result from a subset of the table data. You will use the **db2batch** application to execute the query and report basic elapsed time statistics. You will also review several SQL queries in the db2batch sequence that return important DB2 performance metrics that you will use to compare the row-organized and column-organized table processing.

The file *rowquery1.sql* contains the following statements:

```
set current explain mode yes;

SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance, count(*) as br_trans
   FROM ROWORG.HISTORY AS HISTORY
   WHERE HISTORY.BRANCH_ID between 10 and 35
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;

set current explain mode no;
SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;
SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time,
total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch';
```

The CURRENT EXPLAIN setting will generate the explain data that can be formatted using the **db2exfmt** tool to show estimated processing costs. You will use the file *explain.ddl* to create a new set of explain tables.

You will also collect new table statistics for the column-organized tables.

1. In the terminal session, enter the following commands:
   - **cd $HOME/db2blu**
   - **db2 force application all**
   - **db2 terminate**
   - **db2 activate db db2blu**
   - **db2 connect to db2blu**
   - **db2 -tvf explain.ddl**
   - **db2 -tvf runstats.cmd**

   Next you will use db2batch to execute the first set of SQL statements that access the row organized table ROWORG.HISTORY.

2. In the terminal session, enter the following commands:
   - **db2batch -d db2blu -f rowquery1.sql -i complete -ISO CS | tee rowq1bat.txt**
   - **more rowq1bat.txt**

   Review the db2batch report noting the following information:

   Elapsed time for the query (SQL Statement Number 2):

   For example:

```
* SQL Statement Number 2:

SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance, count(*) as br_trans
   FROM ROWORG.HISTORY AS HISTORY
   WHERE HISTORY.BRANCH_ID between 10 and 35
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;

BRANCH_ID BR_BALANCE                        BR_TRANS
--------- --------------------------------- -----------
       10                     2013883300.00       25939
       11                     2037791600.00       25746
       12                     2515666000.00       29556
       13                     3634933600.00       37235
       14                     2741741500.00       30871

* 26 row(s) fetched, 5 row(s) output.

* Prepare Time is:       0.066990 seconds
* Execute Time is:       0.504817 seconds
* Fetch Time is:         0.000899 seconds
* Elapsed Time is:       0.572706 seconds (complete)
```

The example elapsed time is 0.572706 seconds.

Look at the two queries that return performance metrics using the table function MON_GET_CONNECTION.

For example:

```
* SQL Statement Number 4:

SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS   TOTAL_L_READS
-------------------- -------------------- -------------------- -------------------
-
                   0                 1738                  563
2301

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.022331 seconds
* Execute Time is:       0.008886 seconds
* Fetch Time is:         0.000250 seconds
* Elapsed Time is:       0.031467 seconds (complete)

---------------------------------------------

* SQL Statement Number 5:

SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time,
total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

TOTAL_COL_TIME       TOTAL_COMPILE_TIME   POOL_READ_TIME       TOTAL_CPU_TIME
TOTAL_WAIT_TIME
-------------------- -------------------- -------------------- -------------------
- --------------------
                   0                   93                  156
636674                1017

* 1 row(s) fetched, 1 row(s) output.
```

Note the following statistics from your copy of the report:

TOTAL_L_READS:	(2301 in the sample)
TOTAL_WAIT_TIME:	(1017 in the sample)
POOL_READ_TIME:	(156 in the sample)

Now look at the access plan for the db2exfmt explain report.

3. In the terminal session, enter the following commands:

- **db2exfmt -1 -d db2blu -o exrowq1.txt**
- **more exrowq1.txt**

For example:

```
Access Plan:
-----------
     Total Cost:        1263.87
     Query Degree:            2


            Rows
           RETURN
           (   1)
            Cost
             I/O
             |
             26
           GRPBY
           (   2)
           1263.87
           440.737
             |
             26
           LMTQ
           (   3)
           1263.87
           440.737
             |
             26
           TBSCAN
           (   4)
           1263.85
           440.737
             |
             26
           SORT
           (   5)
```

```
              1263.85
              440.737
                |
                26
              pGRPBY
              (   6)
              1263.85
              440.737
                |
              127625
              FETCH
              (   7)
              1243.72
              440.737
            /---+----\
        127625       490864
        RIDSCN   TABLE: ROWORG
        (   8)       HISTORY
        252.848        Q1
         83.96
          |
        127625
        SORT
        (   9)
        252.847
         83.96
          |
        127625
        IXSCAN
        (  10)
        218.637
         83.96
          |
        490864
    INDEX: ROWORG
        HISTIX1
          Q1
```

Review the db2exfmt report noting the following information:

Total Estimated cost: ___________________ (1263.87 in the sample)

Total I/O cost: _________________________ (440 in the sample)

The access plan uses one index to access the table ROWORG.HISTORY.

You will now run a similar db2batch report using the same SQL query text based on the column-organized table **COLORG.HISTORY** instead of the row-organized table.

4. In the terminal session, enter the following commands:
   - **db2batch -d db2blu -f colquery1.sql -i complete -ISO CS | tee colq1bat.txt**
   - **more colq1bat.txt**

   Review the db2batch report noting the following information:

   Elapsed time for the query (SQL Statement Number 2):

   For example:

```
* SQL Statement Number 2:

SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance, count(*) as br_trans
   FROM COLORG.HISTORY AS HISTORY
   WHERE HISTORY.BRANCH_ID between 10 and 35
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;

BRANCH_ID BR_BALANCE                        BR_TRANS
--------- --------------------------------- -----------
       10                     2013883300.00       25939
       11                     2037791600.00       25746
       12                     2515666000.00       29556
       13                     3634933600.00       37235
       14                     2741741500.00       30871

* 26 row(s) fetched, 5 row(s) output.

* Prepare Time is:       0.033123 seconds
* Execute Time is:       0.083704 seconds
* Fetch Time is:         0.003086 seconds
* Elapsed Time is:       0.119913 seconds (complete)
```

   The example elapsed time is 0.119913 seconds.

   Look at the two queries that return performance metrics using the table function MON_GET_CONNECTION.

For example:

```
* SQL Statement Number 4:

SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS   TOTAL_L_READS
-------------------- -------------------- -------------------- -------------------
-
                 133                  103                  109
345

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.000227 seconds
* Execute Time is:       0.008211 seconds
* Fetch Time is:         0.000295 seconds
* Elapsed Time is:       0.008733 seconds (complete)

---------------------------------------------

* SQL Statement Number 5:

SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time,
total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

TOTAL_COL_TIME       TOTAL_COMPILE_TIME   POOL_READ_TIME       TOTAL_CPU_TIME
TOTAL_WAIT_TIME
-------------------- -------------------- -------------------- -------------------
- --------------------
                 314                   34                   22
94463                 236
```

Note the following statistics from your copy of the report:

TOTAL_L_READS:	(345 in the sample)
TOTAL_WAIT_TIME:	(236 in the sample)
POOL_READ_TIME:	(22 in the sample)

Now look at the access plan for the db2exfmt explain report.

5. In the terminal session, enter the following commands:
   - **db2exfmt -1 -d db2blu -o excolq1.txt**
   - **more excolq1.txt**

   For example:

```
Access Plan:
-----------
     Total Cost:        173.719
     Query Degree:            2


         Rows
        RETURN
        (   1)
         Cost
          I/O
          |
          26
        LMTQ
        (   2)
        173.719
          59
          |
          26
        CTQ
        (   3)
        173.705
          59
          |
          26
        TBSCAN
        (   4)
        173.704
          59
          |
          26
        SORT
        (   5)
        173.703
          59
          |
          26
        GRPBY
        (   6)
        173.694
          59
```

```
             |
           127625
           TBSCAN
           (   7)
           171.703
             59
             |
           490864
     CO-TABLE: COLORG
           HISTORY
             Q1
```

Review the db2exfmt report noting the following information:

Total Estimated cost: ___________________ (173 in the sample)

Total I/O cost: _________________________ (59 in the sample)

The access plan uses a table scan to access the table COLORG.HISTORY.

The estimated I/O costs as well as the actual logical pages read for the query using the column-organized table are less than executing the query using the row-organized table.

One key aspect of BLU Acceleration performance is the ability to perform queries efficiently without standard column based indexes. The next SQL query that you will analyze also accesses a single table to produce a summarized result from a subset of the table data. In this case the subset of data is based on two predicates that are supported by indexes on the table ROWORG.HISTORY. You will use the **db2batch** application to execute the query and report basic elapsed time statistics. You will also include the same SQL queries in the db2batch sequence that return important DB2 performance metrics that you will use to compare the row-organized and column-organized table processing.

You will start by running the SQL query using the row-organized table. The file *rowquery2.sql* contains the following statements:

```
set current explain mode yes;

SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance, count(*) as br_trans
   FROM ROWORG.HISTORY AS HISTORY
   where branch_id between 10 and 30 and teller_id > 800
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;

set current explain mode no;
SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;
SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time,
total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;
```

6. In the terminal session, enter the following commands:

- **db2batch -d db2blu -f rowquery2.sql -i complete -ISO CS | tee rowq2bat.txt**
- **more rowq2bat.txt**

Review the db2batch report noting the following information:

Elapsed time for the query (SQL Statement Number 2):

For example:

```
* SQL Statement Number 2:

SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance, count(*) as br_trans
   FROM ROWORG.HISTORY AS HISTORY
   where branch_id between 10 and 30 and teller_id > 800
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;

BRANCH_ID BR_BALANCE                        BR_TRANS
--------- --------------------------------- -----------
       10                      269211500.00        4261
       11                      674738900.00        6894
       12                      217492500.00        3859
       13                      471080000.00        5716
       14                      216546500.00        4016
```

```
* 21 row(s) fetched, 5 row(s) output.

* Prepare Time is:       0.035656 seconds
* Execute Time is:       0.157717 seconds
* Fetch Time is:         0.007629 seconds
* Elapsed Time is:       0.201002 seconds (complete)
```

The example elapsed time is 0.201002 seconds.

Look at the two queries that return performance metrics using the table function MON_GET_CONNECTION.

For example:

```
* SQL Statement Number 4:

SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS   TOTAL_L_READS
-------------------- -------------------- -------------------- -------------------
-
                   0                 1595                  676
2271

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.000158 seconds
* Execute Time is:       0.008554 seconds
* Fetch Time is:         0.000211 seconds
* Elapsed Time is:       0.008923 seconds (complete)

---------------------------------------------

* SQL Statement Number 5:

SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time,
total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

TOTAL_COL_TIME       TOTAL_COMPILE_TIME   POOL_READ_TIME       TOTAL_CPU_TIME
TOTAL_WAIT_TIME
-------------------- -------------------- -------------------- -------------------
- --------------------
```

```
                   0                   36                     9
309508                 216

* 1 row(s) fetched, 1 row(s) output.
```

Note th+e following statistics from your copy of the report:

TOTAL_L_READS:	(2271 in the sample)
TOTAL_WAIT_TIME:	(216 in the sample)
POOL_READ_TIME:	(9 in the sample)

Now look at the access plan for the db2exfmt explain report.

7. In the terminal session, enter the following commands:

- **db2exfmt -1 -d db2blu -o exrowq2.txt**
- **more exrowq2.txt**

For example:

```
Access Plan:
-----------
     Total Cost:        773.601
     Query Degree:            2


                   Rows
                  RETURN
                  (   1)
                   Cost
                    I/O
                    |
                    21
                  GRPBY
                  (   2)
                   773.6
                  325.281
                    |
                    21
                  LMTQ
                  (   3)
                  773.599
                  325.281
                    |
                    21
                  TBSCAN
                  (   4)
                  773.585
                  325.281
```

```
                    |
                    21
                  SORT
                  (   5)
                  773.585
                  325.281
                    |
                    21
                  pGRPBY
                  (   6)
                  773.581
                  325.281
                    |
                  20616.3
                  FETCH
                  (   7)
                  770.315
                  325.281
                /---+----\
            20616.3      490864
            RIDSCN   TABLE: ROWORG
            (   8)       HISTORY
            298.593        Q1
            95.4434
              |
            20616.3
            SORT
            (   9)
            298.593
            95.4434
              |
            20616.3
            IXAND
            (  10)
            292.347
            95.4434
        /-----+------\
    98172.8          103081
    IXSCAN           IXSCAN
    (  11)           (  12)
    106.518          180.362
    28.7833           66.66
      |                 |
    490864           490864
INDEX: ROWORG    INDEX: ROWORG
    HISTIX2          HISTIX1
      Q1               Q1
```

Review the db2exfmt report noting the following information:

Total Estimated cost: ___________________ (773 in the sample)

Total I/O cost: _________________________ (325 in the sample)

The access plan uses the INDEX ANDING (IXAND) operation to combine two index scan results to access the table ROWORG.HISTORY.

We will run a similar db2batch report using the same SQL query but the column-organized table COLORG.HISTORY will be used instead of the row-organized table.

Notice the db2batch option '-ISO CS' sets the lock isolation option to CS or cursor stability which is supported by BLU Acceleration.

8. In the terminal session, enter the following commands:

- **db2batch -d db2blu -f colquery2.sql -i complete -ISO CS | tee colq2bat.txt**
- **more colq2bat.txt**

Review the db2batch report noting the following information:

Elapsed time for the query (SQL Statement Number 2):

For example:

```
* SQL Statement Number 2:

SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance, count(*) as br_trans
   FROM COLORG.HISTORY AS HISTORY
   where branch_id between 10 and 30 and teller_id > 800
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;

BRANCH_ID BR_BALANCE                        BR_TRANS
--------- --------------------------------- -----------
       10                      269211500.00        4261
       11                      674738900.00        6894
       12                      217492500.00        3859
       13                      471080000.00        5716
       14                      216546500.00        4016

* 21 row(s) fetched, 5 row(s) output.

* Prepare Time is:       0.036034 seconds
* Execute Time is:       0.025701 seconds
* Fetch Time is:         0.025297 seconds
* Elapsed Time is:       0.087032 seconds (complete)
```

The example elapsed time is 0.087032 seconds.

Look at the two queries that return performance metrics using the table function MON_GET_CONNECTION.

For example:

```
* SQL Statement Number 4:

SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS   TOTAL_L_READS
-------------------- -------------------- -------------------- -------------------
-
                  47                  102                  102
251

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.000161 seconds
* Execute Time is:       0.008415 seconds
* Fetch Time is:         0.000212 seconds
* Elapsed Time is:       0.008788 seconds (complete)

---------------------------------------------

* SQL Statement Number 5:

SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time,
total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

TOTAL_COL_TIME       TOTAL_COMPILE_TIME   POOL_READ_TIME       TOTAL_CPU_TIME
TOTAL_WAIT_TIME
-------------------- -------------------- -------------------- -------------------
- --------------------
                  93                   37                    4
59120                   88

* 1 row(s) fetched, 1 row(s) output.
```

Note the following statistics from your copy of the report:

TOTAL_L_READS: ________________________ (251 in the sample)

TOTAL_WAIT_TIME: ________________________ (88 in the sample)

POOL_READ_TIME: ________________________ (4 in the sample)

Now look at the access plan for the db2exfmt explain report.

9. In the terminal session, enter the following commands:

- **db2exfmt -1 -d db2blu -o excolq2.txt**
- **more excolq2.txt**

For example:

```
Access Plan:
-----------
     Total Cost:        191.372
     Query Degree:            2


        Rows
       RETURN
       (   1)
        Cost
         I/O
         |
         21
       LMTQ
       (   2)
       191.372
         65
         |
         21
       CTQ
       (   3)
       191.359
         65
         |
         21
       TBSCAN
       (   4)
       191.358
         65
         |
         21
       SORT
       (   5)
```

```
        191.357
          65
          |
          21
        GRPBY
        (   6)
        191.349
          65
          |
        20616.4
        TBSCAN
        (   7)
        191.027
          65
          |
        490864
  CO-TABLE: COLORG
        HISTORY
          Q1
```

Review the db2exfmt report noting the following information:

Total Estimated cost: _________________ (191 in the sample)

Total I/O cost: _______________________ (65 in the sample)

The access plan uses a table scan to access the table COLORG.HISTORY. Compare these estimated costs to the explain report based on the row-organized table.

Notice that most of the processing, including the SORT operation, is performed in column-organized mode, below the CTQ operation.

We have used two SQL queries to compare performance of row-organized and column-organized tables. For the row-organized table, one or more indexes were used to produce the results.

In the sample results, the column-organized table was able to produce the same results faster with no traditional indexes.

The next SQL query that you will analyze joins two tables to produce a summarized result from a subset of the table data. In this case the subset of data is based on one predicate on the larger table.

The SQL text for the join is:

```
SELECT HISTORY.TELLER_ID, sum(HISTORY.BALANCE) as total_balance,
TELLER.TELLER_NAME , count(*) as transactions
  FROM COLORG.HISTORY AS HISTORY, COLORG.TELLER AS TELLER
   WHERE HISTORY.TELLER_ID = TELLER.TELLER_ID
    and HISTORY.teller_id between 10 and 100
   GROUP BY HISTORY.TELLER_ID, TELLER.TELLER_NAME

   order by 4 desc ;
```

The query joins the HISTORY and TELLER tables using the TELLER_ID column. A subset of the TELLER_ID column values are included in the result.

In order to improve join access plan generation for column-organized tables, we can create NOT ENFORCED referential integrity or unique constraints to better define table data relationships without the need for physical indexes.

Use the file *pkeys.ddl* to define primary key and foreign key constraints on the column-organized tables using the NOT ENFORCED attribute.

The file pkeys.ddl contains the following SQL statements:

```
alter table colorg.acct add primary key (acct_id) not enforced ;
alter table colorg.branch add primary key (branch_id)  not enforced ;
alter table colorg.teller add primary key (teller_id)  not enforced ;

alter table colorg.history add constraint fkeyteller
      foreign key (teller_id) references colorg.teller not enforced ;

alter table colorg.history add constraint fkeybranch
      foreign key (branch_id) references colorg.branch not enforced ;

alter table colorg.history add constraint fkeyacct
      foreign key (acct_id) references colorg.acct not enforced ;
```

10. In the terminal session, enter the following commands:
    - **db2 connect to db2blu**
    - **db2 -tvf pkeys.ddl**

    You will start by running the SQL query using the row-organized table. The file *rowquery3.sql* contains the following statements:

```
SELECT HISTORY.TELLER_ID, sum(HISTORY.BALANCE) as total_balance,
TELLER.TELLER_NAME , count(*) as transactions
  FROM ROWORG.HISTORY AS HISTORY, ROWORG.TELLER AS TELLER
   WHERE HISTORY.TELLER_ID = TELLER.TELLER_ID
    and HISTORY.teller_id between 10 and 100
   GROUP BY HISTORY.TELLER_ID, TELLER.TELLER_NAME
   order by 4 desc ;

set current explain mode no;
SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;
SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time,
total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;
```

11. In the terminal session, enter the following commands:
    - **db2batch -d db2blu -f rowquery3.sql -i complete -ISO CS | tee rowq3bat.txt**
    - **more rowq3bat.txt**

    Review the db2batch report noting the following information:

    Elapsed time for the query (SQL Statement Number 2):

    For example:

```
* SQL Statement Number 2:

SELECT HISTORY.TELLER_ID, sum(HISTORY.BALANCE) as total_balance,
TELLER.TELLER_NAME , count(*) as transactions
  FROM ROWORG.HISTORY AS HISTORY, ROWORG.TELLER AS TELLER
   WHERE HISTORY.TELLER_ID = TELLER.TELLER_ID
    and HISTORY.teller_id between 10 and 100
   GROUP BY HISTORY.TELLER_ID, TELLER.TELLER_NAME
   order by 4 desc ;

TELLER_ID TOTAL_BALANCE                         TELLER_NAME          TRANSACTIONS
```

```
--------- ------------------------------ ------------------ -----------
       30                   270329400.00 <--20 BYTE STRING-->         1891
       96                   248278700.00 <--20 BYTE STRING-->         1858
       55                   236779400.00 <--20 BYTE STRING-->         1789
       49                   243847100.00 <--20 BYTE STRING-->         1774
       93                   216283800.00 <--20 BYTE STRING-->         1722

* 91 row(s) fetched, 5 row(s) output.

* Prepare Time is:       0.807181 seconds
* Execute Time is:       0.047823 seconds
* Fetch Time is:         0.001857 seconds
* Elapsed Time is:       0.856861 seconds (complete)
```

The example elapsed time is 0.856861 seconds.

Look at the two queries that return performance metrics using the table function MON_GET_CONNECTION.

For example:

```
* SQL Statement Number 4:

SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS   TOTAL_L_READS
-------------------- -------------------- -------------------- -------------------
-
                   0                 2733                  620
3353

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.000215 seconds
* Execute Time is:       0.008266 seconds
* Fetch Time is:         0.000292 seconds
* Elapsed Time is:       0.008773 seconds (complete)

---------------------------------------------

* SQL Statement Number 5:

SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time,
total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

TOTAL_COL_TIME       TOTAL_COMPILE_TIME   POOL_READ_TIME       TOTAL_CPU_TIME
TOTAL_WAIT_TIME
-------------------- -------------------- -------------------- -------------------
- --------------------
                   0                  808                   78
758560                  213

* 1 row(s) fetched, 1 row(s) output.
```

Note the following statistics from your copy of the report:

TOTAL_L_READS:	(3353 in the sample)
TOTAL_WAIT_TIME:	(213 in the sample)
POOL_READ_TIME:	(78 in the sample)

Now look at the access plan for the db2exfmt explain report.

12. In the terminal session, enter the following commands:

- **db2exfmt -1 -d db2blu -o exrowq3.txt**
- **more exrowq3.txt**

For example:

```
Access Plan:
-----------
     Total Cost:        1792.33
     Query Degree:            2


                          Rows
                         RETURN
                         (   1)
                          Cost
                           I/O
                           |
                         90.283
                         TBSCAN
                         (   2)
                         1792.33
                         836.581
                           |
                         90.283
                         SORT
                         (   3)
                         1792.32
                         836.581
```

```
                          |
                        90.283
                        GRPBY
                        (   4)
                        1792.3
                        836.581
                          |
                        90.283
                        LMTQ
                        (   5)
                        1792.3
                        836.581
                          |
                        90.283
                        TBSCAN
                        (   6)
                        1792.27
                        836.581
                          |
                        90.283
                        SORT
                        (   7)
                        1792.27
                        836.581
                          |
                        90.7522
                        pGRPBY
                        (   8)
                        1792.25
                        836.581
                          |
+                        44316.7
                        ^HSJOIN
                        (   9)
                        1783.16
                        836.581
              /-----------+-----------\
        45818.8                      90.283
        FETCH                        FETCH
        (  10)                       (  14)
        1774.93                      7.10783
        835.581                         1
      /---+----\                  /----+----\
  45818.8     490864        90.283         1000
  RIDSCN   TABLE: ROWORG    IXSCAN    TABLE: ROWORG
  (  11)       HISTORY      (  15)        TELLER
  112.731        Q2        0.0503556        Q1
```

```
      12.9747                       0
        |                          |
      45818.8                    1000
      SORT                  INDEX: SYSIBM
      (  12)              SQL170503161646610
      112.731                      Q1
      12.9747
        |
      45818.8
      IXSCAN
      (  13)
      101.68
      12.9747
        |
      490864
  INDEX: ROWORG
      HISTIX2
        Q2
```

Review the db2exfmt report noting the following information:

Total Estimated cost:	(1792 in the sample)
Total I/O cost:	(836 in the sample)

The access plan uses the indexes on both tables. The join processing uses a hash join method.

We will run a similar db2batch report using the same SQL query but the column-organized tables instead of the row-organized tables.

13. In the terminal session, enter the following commands:

- **db2batch -d db2blu -f colquery3.sql -i complete -ISO CS | tee colq3bat.txt**
- **more colq3bat.txt**

Review the db2batch report noting the following information:

Elapsed time for the query (SQL Statement Number 2):

For example:

```
* SQL Statement Number 2:

SELECT HISTORY.TELLER_ID, sum(HISTORY.BALANCE) as total_balance,
TELLER.TELLER_NAME , count(*) as transactions
  FROM COLORG.HISTORY AS HISTORY, COLORG.TELLER AS TELLER
   WHERE HISTORY.TELLER_ID = TELLER.TELLER_ID
    and HISTORY.teller_id between 10 and 100
   GROUP BY HISTORY.TELLER_ID, TELLER.TELLER_NAME
   order by 4 desc ;

TELLER_ID TOTAL_BALANCE                     TELLER_NAME          TRANSACTIONS
--------- --------------------------------- -------------------- ------------
       30                      270329400.00 <--20 BYTE STRING-->         1891
       96                      248278700.00 <--20 BYTE STRING-->         1858
       55                      236779400.00 <--20 BYTE STRING-->         1789
       49                      243847100.00 <--20 BYTE STRING-->         1774
       93                      216283800.00 <--20 BYTE STRING-->         1722

* 91 row(s) fetched, 5 row(s) output.

* Prepare Time is:       0.054485 seconds
* Execute Time is:       0.157474 seconds
* Fetch Time is:         0.068527 seconds
* Elapsed Time is:       0.280486 seconds (complete)
```

The example elapsed time is 0.280486 seconds.

Look at the two queries that return performance metrics using the table function MON_GET_CONNECTION.

For example:

```
* SQL Statement Number 4:

SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS   TOTAL_L_READS
-------------------- -------------------- -------------------- -------------------
-
                  46                  197                  149
392

* 1 row(s) fetched, 1 row(s) output.
* Prepare Time is:       0.032008 seconds
* Execute Time is:       0.066819 seconds
* Fetch Time is:         0.000480 seconds
* Elapsed Time is:       0.099307 seconds (complete)

---------------------------------------------

* SQL Statement Number 5:

SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time,
total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

TOTAL_COL_TIME       TOTAL_COMPILE_TIME   POOL_READ_TIME       TOTAL_CPU_TIME
TOTAL_WAIT_TIME
-------------------- -------------------- -------------------- -------------------
- --------------------
                1049                   92                  100
161485                  567

* 1 row(s) fetched, 1 row(s) output.
```

Note the following statistics from your copy of the report:

TOTAL_L_READS:	(392 in the sample)
TOTAL_WAIT_TIME:	(567 in the sample)
POOL_READ_TIME:	(92 in the sample)

Now look at the access plan for the db2exfmt explain report.

14. In the terminal session, enter the following commands:

- **db2exfmt -1 -d db2blu -o excolq3.txt**
- **more excolq3.txt**

For example:

```
Access Plan:
-----------
     Total Cost:        248.597
     Query Degree:            2


               Rows
              RETURN
              (   1)
               Cost
                I/O
```

```
                 |
               91.0014
               LMTQ
               (   2)
               248.597
                 70
                 |
               91.0014
               CTQ
               (   3)
               248.576
                 70
                 |
               91.0014
               TBSCAN
               (   4)
               248.574
                 70
                 |
               91.0014
               SORT
               (   5)
               248.571
                 70
                 |
               91.0014
               ^HSJOIN
               (   6)
               248.538
                 70
            /----+----\
        1000         90.8194
       TBSCAN        GRPBY
       (   7)        (   8)
       77.5445       170.983
         11            59
         |             |
        1000         44669.3
 CO-TABLE: COLORG    TBSCAN
       TELLER        (   9)
         Q1          170.285
                       59
                       |
                     490864
                CO-TABLE: COLORG
                     HISTORY
                       Q2
```

Review the db2exfmt report noting the following information:

Total Estimated cost: _________________ (248 in the sample)

Total I/O cost: _______________________ (70 in the sample)

The access plan uses table scans for the two tables. The join processing uses a hash join method.

For the two table join query, the column-organized tables showed lower estimated and actual execution costs and an improved query elapsed time compared to the row-organized tables with standard indexes.

We could have performed some additional tuning with the row-organized tables, looking for better indexes or table organizations, but the column-organized tables may offer a simple solution that performs well and does not require the additional performance analysis.

Task 3. Implement a primary key index for a column-organized table to improve single row update processing.

You will now look at the performance results for running a series of single row updates using a column-organized table. The INGEST utility will be used to perform the UPDATE processing based on a series of transaction records.

The table COLORG.ACCT has a unique key based on the ACCT_ID column. The INGEST utility will be used to perform a batch of updates, each making a change to a single row. The table COLORG.ACCT has a NOT ENFORCED primary key constraint, so no physical index is maintained on the ACCT_ID column. Each UPDATE processed will generate a table scan to find all matching data rows.

You will start by running the INGEST command in the file *ingest_update1.ddl*, which also includes a SQL query that shows table access statistics using MON_GET_TABLE. We will deactivate the database to be able to isolate the table statistics for the INGEST processing.

The file ingest_update1.txt contains the following statements:

```
INGEST SET COMMIT_COUNT 1000 ;

ingest from file transfile.del format delimited
(  $key_acct integer external,
   $data_bal decimal (15,2) external ) messages colingest1.txt restart off
 update colorg.acct
 set balance = $data_bal
 where acct_id = $key_acct ;

select
varchar(tabname,12) as table,
varchar(tabschema,12) as schema,
rows_read, rows_updated, rows_inserted, rows_deleted,
table_scans,
object_data_l_reads as data_l_reads,
object_col_l_reads as column_l_reads

from table(mon_get_table(NULL,NULL,-2)) as t1
where tabschema IN ('COLORG','ROWORG')
 or (tabschema = 'SYSIBM' and tabname like 'SYN%')
order by tabname ;
```

1. In the terminal session, enter the following commands:
   - **cd $HOME/db2blu**
   - **db2 force application all**
   - **db2 terminate**
   - **db2 deactivate db db2blu**
   - **db2 connect to db2blu**
   - **db2 -tvf ingest_update1.ddl | tee colupdate1.txt**
   - **more colupdate1.txt**

The output from this command will be similar to the following:

```
INGEST SET COMMIT_COUNT 1000
DB20000I  The INGEST SET command completed successfully.

ingest from file transfile.del format delimited (  $key_acct integer external,
$data_bal decimal (15,2) external ) messages colingest1.txt restart off update
colorg.acct set balance = $data_bal where acct_id = $key_acct

Number of rows read         = 4176
Number of rows inserted     = 0
Number of rows rejected     = 0

SQL2901I  The ingest utility completed at timestamp "05/04/2017
15:28:31.696722". Number of errors: "0". Number of warnings: "1".    Message
file: "colingest1.txt".

select varchar(tabname,12) as table, varchar(tabschema,12) as schema, rows_read,
rows_updated, rows_inserted, rows_deleted, table_scans, object_data_l_reads as
data_l_reads, object_col_l_reads as column_l_reads from
table(mon_get_table(NULL,NULL,-2)) as t1 where tabschema IN ('COLORG','ROWORG') or
(tabschema = 'SYSIBM' and tabname like 'SYN%') order by tabname

TABLE        SCHEMA       ROWS_READ            ROWS_UPDATED         ROWS_INSERTED
ROWS_DELETED         TABLE_SCANS          DATA_L_READS         COLUMN_L_READS
------------ ------------ -------------------- -------------------- -------------
------ -------------------- -------------------- -------------------- -----------
--------
ACCT         COLORG                   44446802                    0
4176                4176                    0                   44
162499
SYN170503160 SYSIBM                    4086416                    0
0                   0                 4176                    3
20964
SQL0445W  Value "SYN170503160255577775_ACCT" has been truncated.
SQLSTATE=01004
```

Notice that the UPDATE processing by INGEST is reported in the MON_GET_TABLE query results as insert and delete activity for the column-organized ACCT table, rather than ROWS_UPDATED. The performance statistics show large numbers of logical reads and rows read. The access to the related synopsis table are also shown.

DB2 can use the physical index associated with an enforced primary key on a column organized table to improve access to single rows.

You will need to drop the NOT ENFORCED primary key for the COLORG.ACCT table and create an ENFORCED primary key based on the ACCT_ID column. You can run a second INGEST command to perform the same series of UPDATE processing once the primary key index is created.

2. In the terminal session, enter the following commands:
   - **db2 connect to db2blu**
   - **db2 -tvf acctprimary.ddl**

   The output from this command will be similar to the following:

```
alter table colorg.acct drop primary key
DB20000I  The SQL command completed successfully.

alter table colorg.acct add primary key (acct_id) enforced
DB20000I  The SQL command completed successfully.
```

3. In the terminal session, enter the following commands:
   - **db2 runstats on table colorg.acct and indexes all**
   - **db2 terminate**
   - **db2 deactivate db db2blu**
   - **db2 connect to db2blu**
   - **db2 -tvf ingest_update2.ddl | tee colupdate2.txt**
   - **more colupdate2.txt**

   The output from this command will be similar to the following:

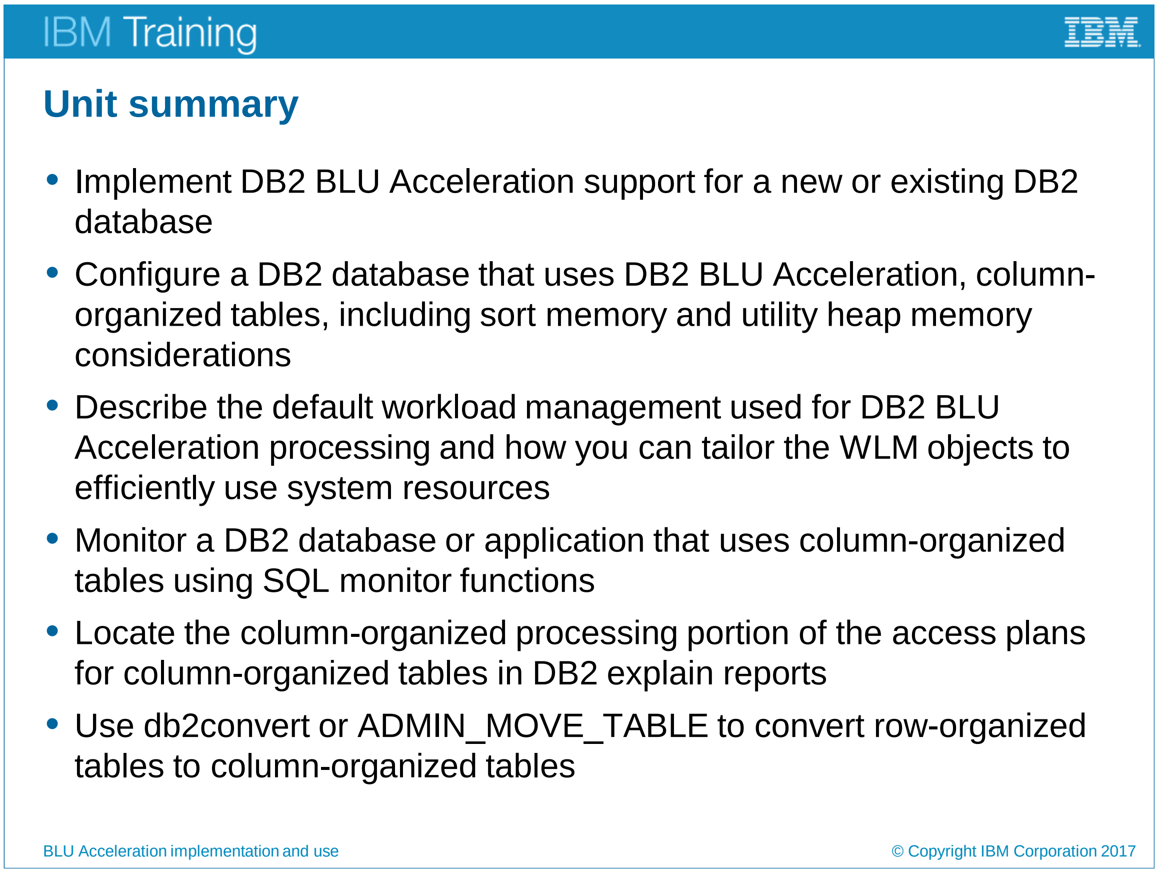
```
INGEST SET COMMIT_COUNT 1000
DB20000I  The INGEST SET command completed successfully.

ingest from file transfile.del format delimited (  $key_acct integer external,
$data_bal decimal (15,2) external ) messages colingest2.txt restart off update
colorg.acct set balance = $data_bal where acct_id = $key_acct

Number of rows read         = 4176
Number of rows inserted     = 0
Number of rows rejected     = 0

SQL2980I  The ingest utility completed successfully at timestamp "05/04/2017
15:33:39.119709"

select varchar(tabname,12) as table, varchar(tabschema,12) as schema, rows_read,
rows_updated, rows_inserted, rows_deleted, table_scans, object_data_l_reads as
data_l_reads, object_col_l_reads as column_l_reads from
table(mon_get_table(NULL,NULL,-2)) as t1 where tabschema IN ('COLORG','ROWORG') or
(tabschema = 'SYSIBM' and tabname like 'SYN%') order by tabname
```

```
TABLE        SCHEMA       ROWS_READ            ROWS_UPDATED         ROWS_INSERTED
ROWS_DELETED         TABLE_SCANS          DATA_L_READS         COLUMN_L_READS
------------ ------------ -------------------- -------------------- --------------
------ -------------------- -------------------- -------------------- ------------
--------
ACCT         COLORG                          0                    0
4176                 4176                    0                   44
23784
SYN170503160 SYSIBM                          0                    0
0                    0                    0                    3
84
SQL0445W  Value "SYN170503160255577775_ACCT" has been truncated.
SQLSTATE=01004
```

The performance statistics returned by the MON_GET_TABLE function for the second INGEST execution with the primary key index defined are different from the first INGEST processing.

- The ROWS_READ for the ACCT table is zero rather than a large number.
- The number of logical reads for column-organized object pages, COLUMN_L_READS, is much lower than the first execution of the INGEST processing.
- Access statistics for the synopsis table are much lower for this second test run.

You may have noticed a significant reduction in the elapsed time for the second execution of the INGEST command.



IBM Training

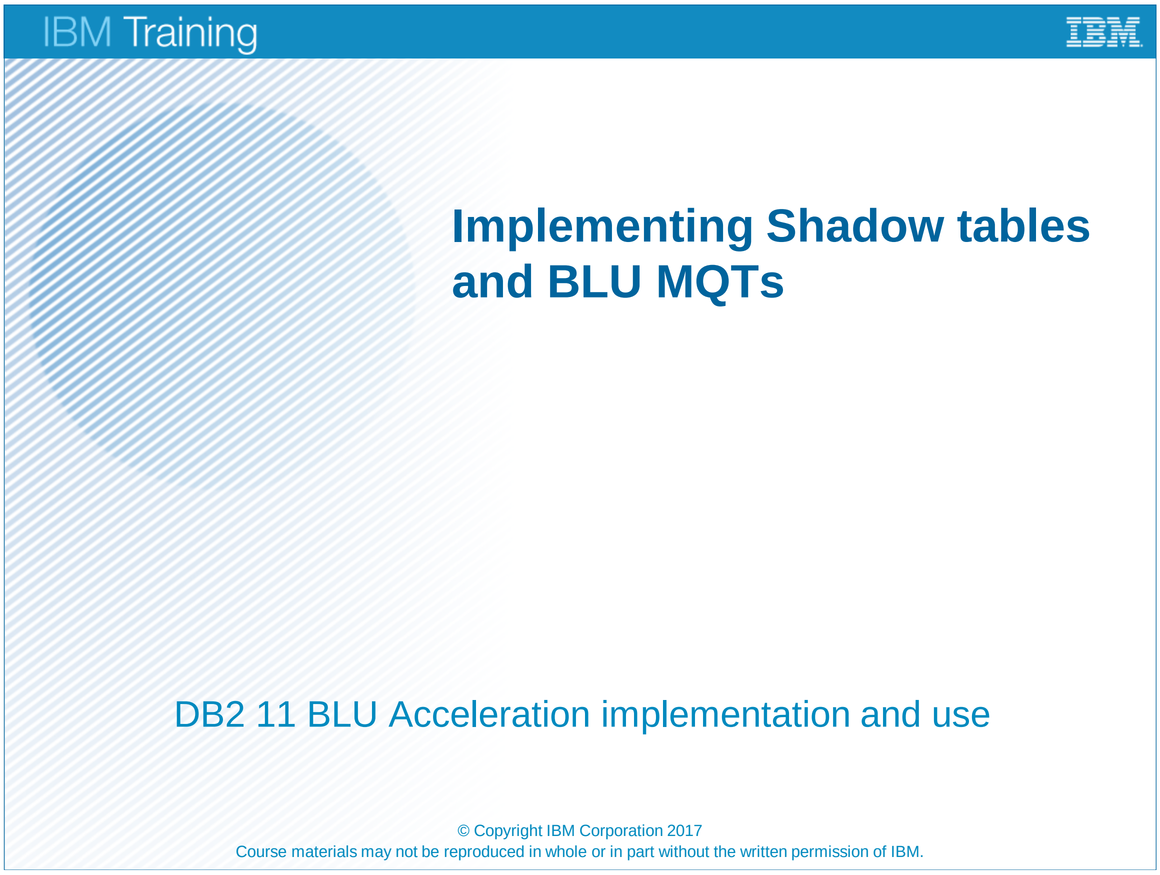
Unit summary

- Implement DB2 BLU Acceleration support for a new or existing DB2 database
- Configure a DB2 database that uses DB2 BLU Acceleration, column-organized tables, including sort memory and utility heap memory considerations
- Describe the default workload management used for DB2 BLU Acceleration processing and how you can tailor the WLM objects to efficiently use system resources
- Monitor a DB2 database or application that uses column-organized tables using SQL monitor functions
- Locate the column-organized processing portion of the access plans for column-organized tables in DB2 explain reports
- Use db2convert or ADMIN_MOVE_TABLE to convert row-organized tables to column-organized tables

BLU Acceleration implementation and use

© Copyright IBM Corporation 2017

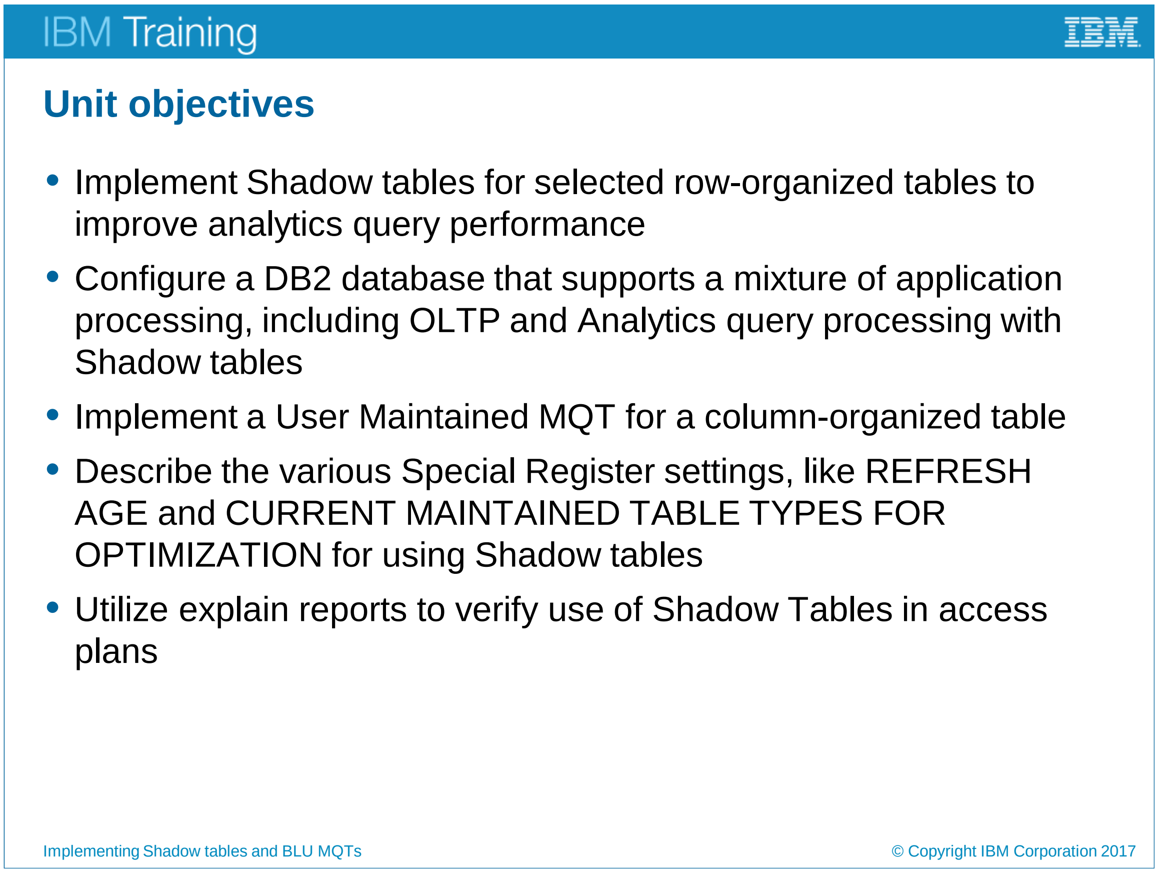
Unit summary



IBM Training

Implementing Shadow tables and BLU MQTs

DB2 11 BLU Acceleration implementation and use



IBM Training

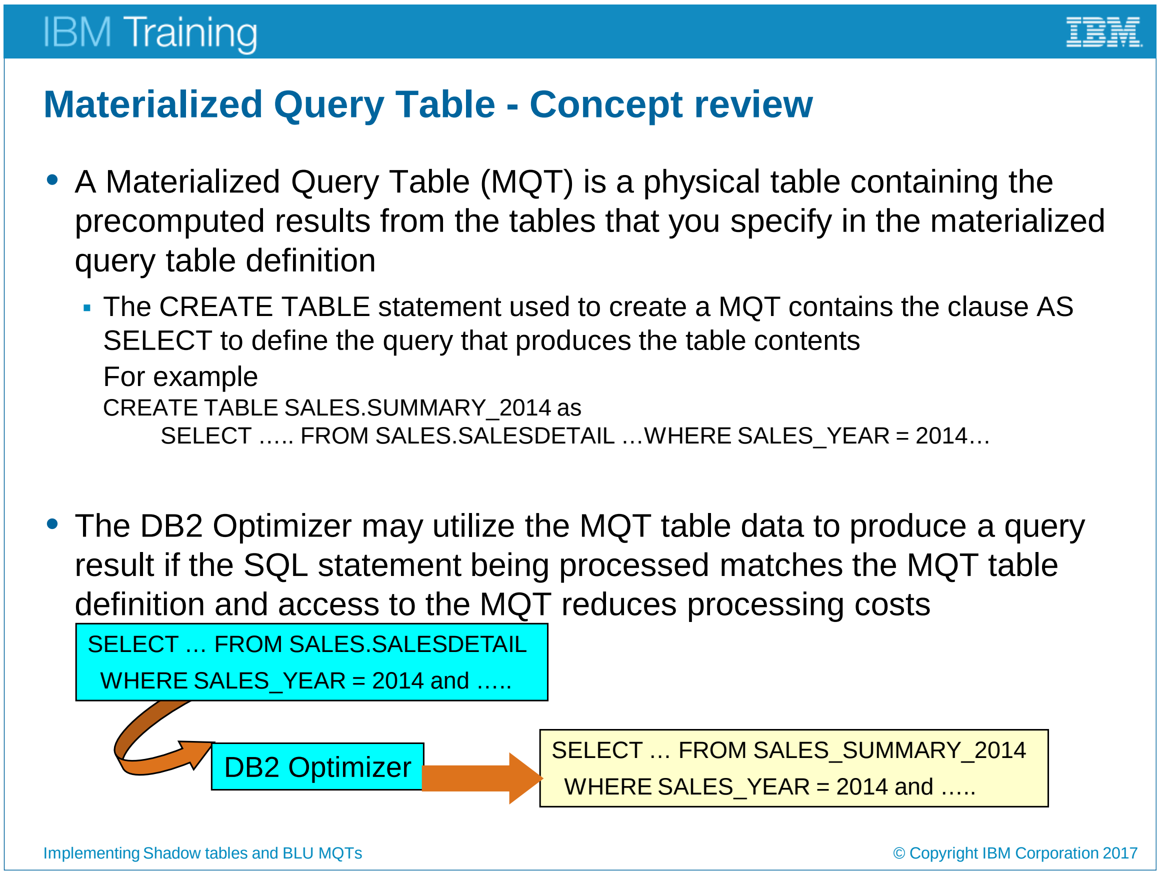
Unit objectives

- Implement Shadow tables for selected row-organized tables to improve analytics query performance
- Configure a DB2 database that supports a mixture of application processing, including OLTP and Analytics query processing with Shadow tables
- Implement a User Maintained MQT for a column-organized table
- Describe the various Special Register settings, like REFRESH AGE and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION for using Shadow tables
- Utilize explain reports to verify use of Shadow Tables in access plans

Implementing Shadow tables and BLU MQTs

© Copyright IBM Corporation 2017

Unit objectives



IBM Training

Materialized Query Table - Concept review

- A Materialized Query Table (MQT) is a physical table containing the precomputed results from the tables that you specify in the materialized query table definition
  - The CREATE TABLE statement used to create a MQT contains the clause AS SELECT to define the query that produces the table contents
    For example

```
CREATE TABLE SALES.SUMMARY_2014 as
     SELECT ….. FROM SALES.SALESDETAIL …WHERE SALES_YEAR = 2014…
```

- The DB2 Optimizer may utilize the MQT table data to produce a query result if the SQL statement being processed matches the MQT table definition and access to the MQT reduces processing costs

SELECT … FROM SALES.SALESDETAIL
WHERE SALES_YEAR = 2014 and …..
DB2 Optimizer
SELECT … FROM SALES_SUMMARY_2014
WHERE SALES_YEAR = 2014 and …..

Implementing Shadow tables and BLU MQTs

© Copyright IBM Corporation 2017

Materialized Query Table - Concept review

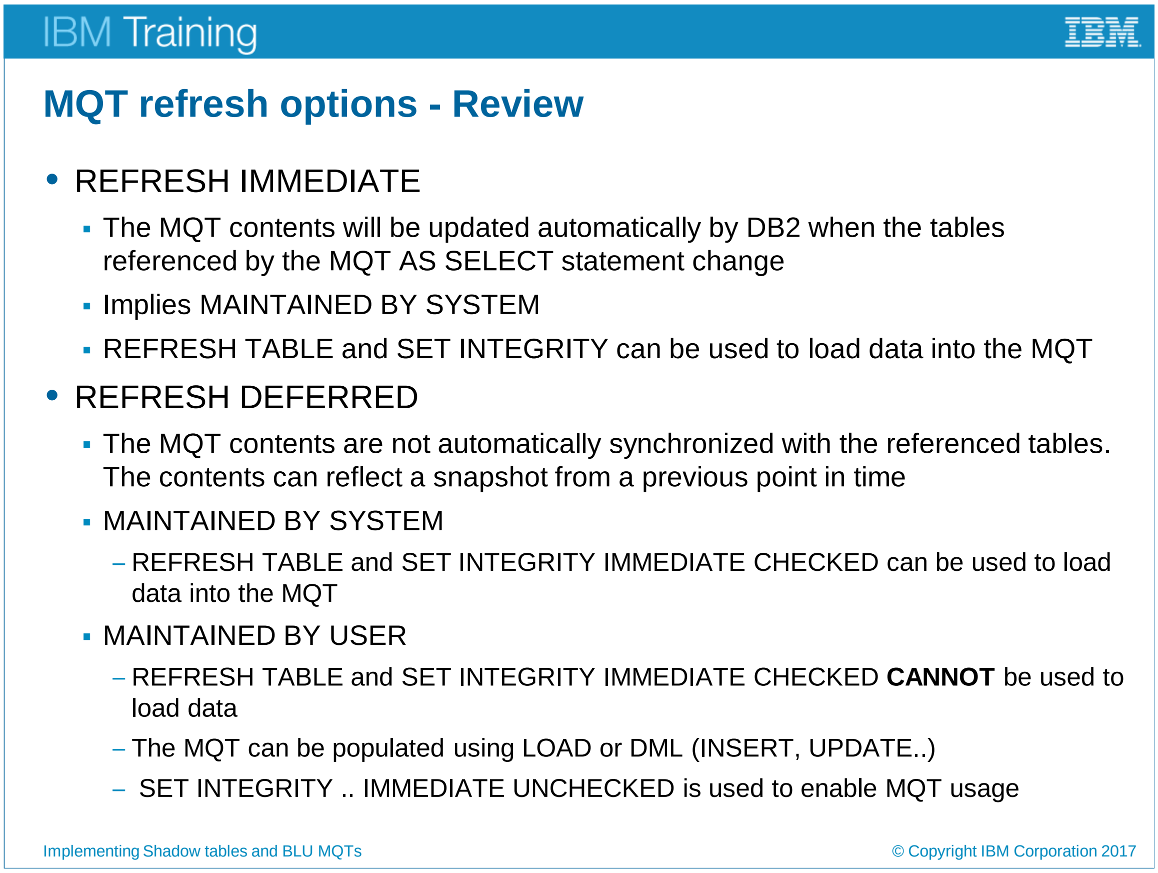
Materialized query tables (MQTs) are a powerful way to improve response time for complex analytical queries because their data consists of precomputed results from the tables that you specify in the materialized query table definitions.

MQTs can help improve response time particularly for queries that use one or more of the following types of data:

- Aggregate data over one or more dimensions
- Joins and aggregate data over a group of tables
- Data from a commonly accessed subset of data
- Repartitioned data from a table, or part of a table, in a partitioned database environment

The larger the base tables, the more significant are the potential improvements in response time when you use MQTs.

During the query rewrite phase, the optimizer determines whether to use an available MQT in place of accessing the referenced base tables directly. If an MQT is used, you need access privileges on the base tables, not the MQT, and the explain facility can provide information about which MQT was selected.



IBM Training

MQT refresh options - Review

- REFRESH IMMEDIATE
  - The MQT contents will be updated automatically by DB2 when the tables referenced by the MQT AS SELECT statement change
  - Implies MAINTAINED BY SYSTEM
  - REFRESH TABLE and SET INTEGRITY can be used to load data into the MQT
- REFRESH DEFERRED
  - The MQT contents are not automatically synchronized with the referenced tables. The contents can reflect a snapshot from a previous point in time
  - MAINTAINED BY SYSTEM
    - REFRESH TABLE and SET INTEGRITY IMMEDIATE CHECKED can be used to load data into the MQT
  - MAINTAINED BY USER
    - REFRESH TABLE and SET INTEGRITY IMMEDIATE CHECKED **CANNOT** be used to load data
    - The MQT can be populated using LOAD or DML (INSERT, UPDATE..)
    - SET INTEGRITY .. IMMEDIATE UNCHECKED is used to enable MQT usage

Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

MQT refresh options - Review

DB2 supports different refresh modes for materialized query tables:

- REFRESH IMMEDIATE

  If a MQT table is defined as REFRESH IMMEDIATE, The changes made to the underlying tables as part of a DELETE, INSERT, or UPDATE are cascaded to the materialized query table. In this case, the content of the table, at any point-in-time, is the same as if the specified subselect is processed. Materialized query tables defined with this attribute do not allow INSERT, UPDATE, or DELETE statements

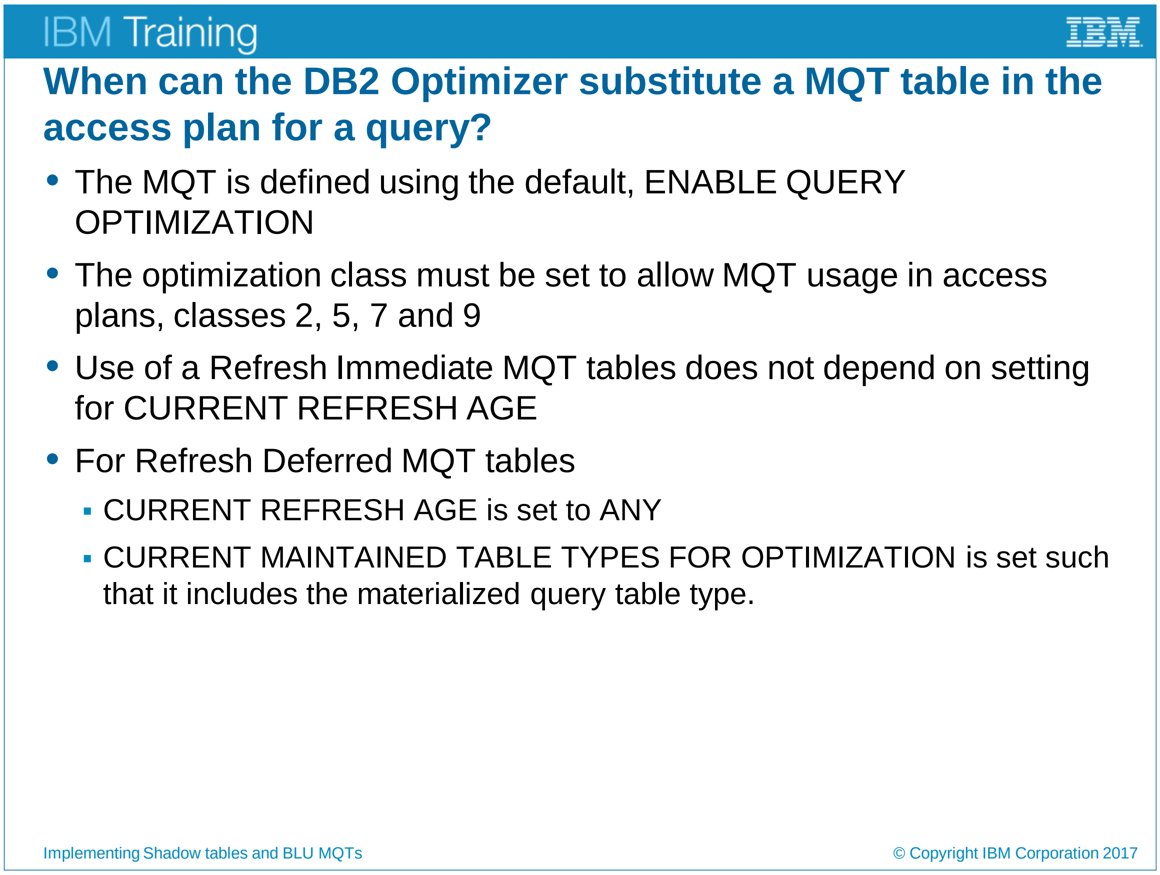
  The REFRESH TABLE and SET INTEGRITY with IMMEDIATE CHECKED statements can be used to refresh the MQT contents.

- REFRESH DEFERRED

  A MQT table can be defined as REFRESH DEFERRED, The changes made to the underlying tables are not cascaded to the materialized query table.

  For system maintained MQT tables, the data in the table can be refreshed at any time using the REFRESH TABLE statement. The data in the table only reflects the result of the query as a snapshot at the time the REFRESH TABLE statement is processed. System-maintained materialized query tables defined with this attribute do not allow INSERT, UPDATE, or DELETE statements.

User-maintained materialized query tables defined with this attribute do allow INSERT, UPDATE, or DELETE statements and can also be populated using a LOAD utility. The SET INTEGRITY statement, with the IMMEDIATE UNCHECKED option is used to let DB2 know that the MQT contents are valid.


IBM Training

When can the DB2 Optimizer substitute a MQT table in the access plan for a query?

- The MQT is defined using the default, ENABLE QUERY OPTIMIZATION
- The optimization class must be set to allow MQT usage in access plans, classes 2, 5, 7 and 9
- Use of a Refresh Immediate MQT tables does not depend on setting for CURRENT REFRESH AGE
- For Refresh Deferred MQT tables
  - CURRENT REFRESH AGE is set to ANY
  - CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION is set such that it includes the materialized query table type.

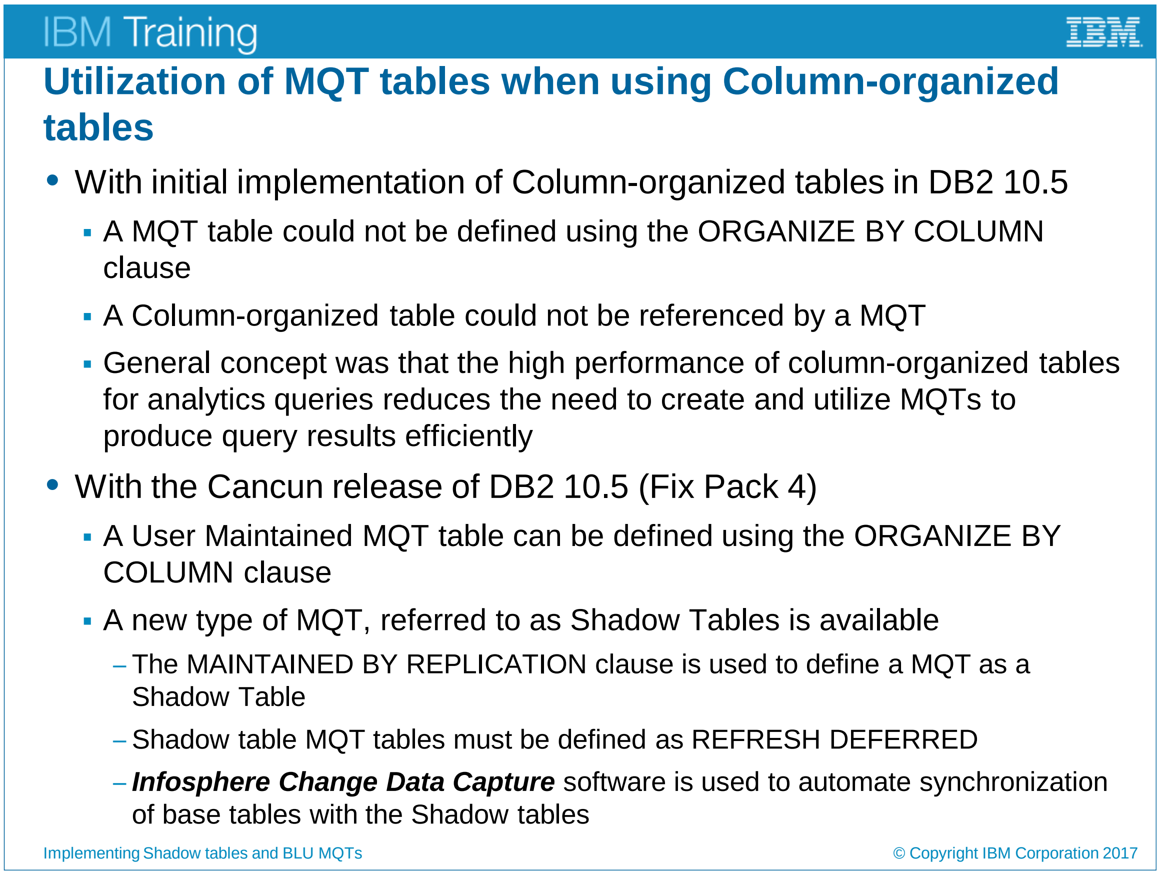
Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

When can the DB2 Optimizer substitute a MQT table in the access plan for a query?

A REFRESH IMMEDIATE materialized query table defined with ENABLE QUERY OPTIMIZATION is always considered for optimization if CURRENT QUERY OPTIMIZATION is set to 2 or a value greater than or equal to 5.

A REFRESH DEFERRED materialized query table defined with ENABLE QUERY OPTIMIZATION can be used to optimize the processing of queries if each of the following conditions is true:

- CURRENT REFRESH AGE is set to ANY.
- CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION is set such that it includes the materialized query table type.
- CURRENT QUERY OPTIMIZATION is set to 2 or a value greater than or equal to 5.



IBM Training

Utilization of MQT tables when using Column-organized tables

- With initial implementation of Column-organized tables in DB2 10.5
  - A MQT table could not be defined using the ORGANIZE BY COLUMN clause
  - A Column-organized table could not be referenced by a MQT
  - General concept was that the high performance of column-organized tables for analytics queries reduces the need to create and utilize MQTs to produce query results efficiently
- With the Cancun release of DB2 10.5 (Fix Pack 4)
  - A User Maintained MQT table can be defined using the ORGANIZE BY COLUMN clause
  - A new type of MQT, referred to as Shadow Tables is available
    - The MAINTAINED BY REPLICATION clause is used to define a MQT as a Shadow Table
    - Shadow table MQT tables must be defined as REFRESH DEFERRED
    - ***Infosphere Change Data Capture*** software is used to automate synchronization of base tables with the Shadow tables

Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

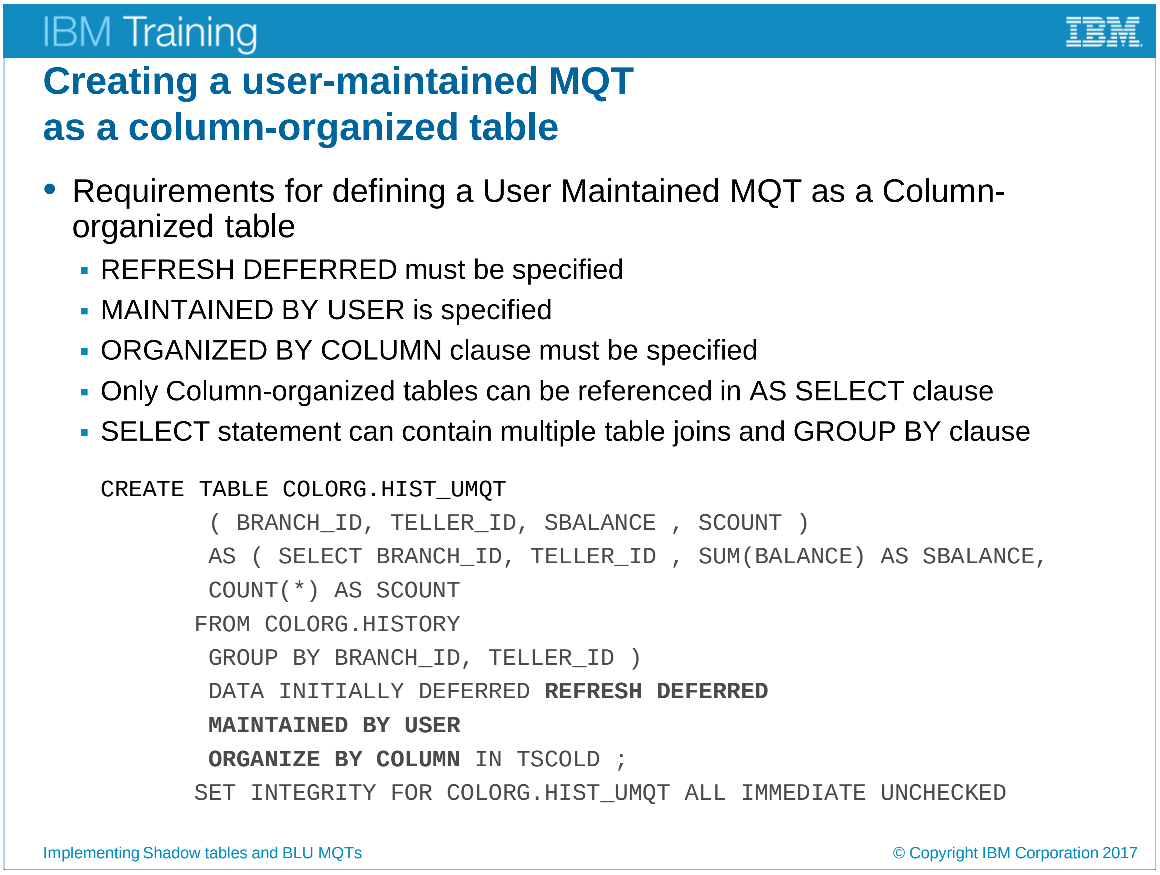
Utilization of MQT tables when using Column-organized tables

With the introduction of Column-organized tables in DB2 10.5, there was a restriction that a MQT table could not be created using the ORGANIZE BY COLUMN clause. There was another restriction that a MQT table could not reference a base table that was column-organized. Since the use of MQT tables is to improve query performance and the use of BLU Acceleration with column-organized tables was designed to dramatically improve performance, the need for MQT tables was considered less critical.

Starting with Fix Pack 4 of DB2 10.5, the following support was added:

- Creation of column-organized user-maintained materialized query tables (MQTs). This enhancement is particularly useful if you are upgrading your DB2 server to Version 10.5 and have existing MQTs. Help reduce upgrade costs by converting your MQTs into column-organized user-maintained MQTs that are eligible to match queries that contain a mix of column-organized and row-organized tables.

- Creation of shadow tables, which is a column-organized copy of a row-organized table that includes all columns or a subset of columns. Shadow tables are implemented as materialized query tables (MQTs) that are maintained by replication. Using shadow tables, you can get the performance benefits of BLU Acceleration for analytic queries in an online transaction processing (OLTP) environment. Analytical queries against row-organized tables are automatically routed to shadow tables if the replication latency falls within a user-defined limit. Shadow tables are maintained by IBM® InfoSphere Change Data Capture for DB2 (InfoSphere CDC), a component of the InfoSphere Data Replication product. InfoSphere CDC asynchronously replicates DML statements that are applied on the source table to the shadow table. By default, all applications access the source tables. Queries are automatically routed to the source table (row-organized) or the shadow table (column-organized copy of the source table) by using a latency-based algorithm that prevents applications from accessing the shadow table when the latency is beyond the user-defined limit.



IBM Training

Creating a user-maintained MQT as a column-organized table

- Requirements for defining a User Maintained MQT as a Column-organized table
  - REFRESH DEFERRED must be specified
  - MAINTAINED BY USER is specified
  - ORGANIZED BY COLUMN clause must be specified
  - Only Column-organized tables can be referenced in AS SELECT clause
  - SELECT statement can contain multiple table joins and GROUP BY clause

```
CREATE TABLE COLORG.HIST_UMQT
        ( BRANCH_ID, TELLER_ID, SBALANCE , SCOUNT )
        AS ( SELECT BRANCH_ID, TELLER_ID , SUM(BALANCE) AS SBALANCE,
        COUNT(*) AS SCOUNT
       FROM COLORG.HISTORY
        GROUP BY BRANCH_ID, TELLER_ID )
        DATA INITIALLY DEFERRED REFRESH DEFERRED
        MAINTAINED BY USER
        ORGANIZE BY COLUMN IN TSCOLD ;
       SET INTEGRITY FOR COLORG.HIST_UMQT ALL IMMEDIATE UNCHECKED
```

Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

Creating a user-maintained MQT as a column-organized table

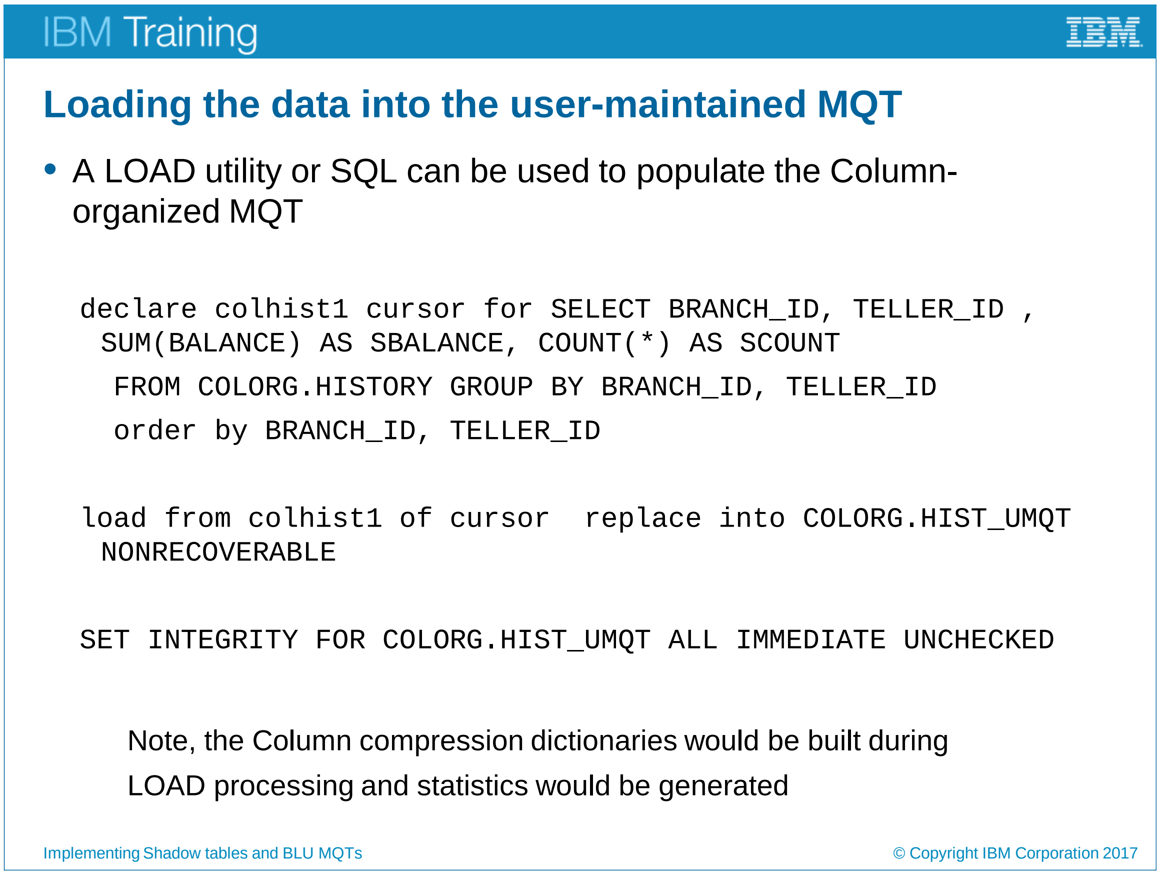
Beginning with DB2 10.5 Fix Pack 4, you can create user maintained MQT tables that are column-organized.

The CREATE TABLE statement used to define a column-organized MQT must have the following options included:

- You must specify the ORGANIZE BY COLUMN clause when creating a column-organized MQT
- MAINTAINED BY USER must be specified, as system maintained column-organized MQT tables are not supported.
- REFRESH DEFERRED must be specified. since REFRESH IMMEDIATE is unsupported
- The referenced source tables must be column-organized.

User maintained column-organized MQT tables can include joins and GROUP BY clauses.

The example shows a column-organized MQT table definition that is based on a SQL statement that contains summarized results from a single source table.


IBM Training

Loading the data into the user-maintained MQT

- A LOAD utility or SQL can be used to populate the Column-organized MQT

```
declare colhist1 cursor for SELECT BRANCH_ID, TELLER_ID ,
 SUM(BALANCE) AS SBALANCE, COUNT(*) AS SCOUNT
  FROM COLORG.HISTORY GROUP BY BRANCH_ID, TELLER_ID
  order by BRANCH_ID, TELLER_ID

load from colhist1 of cursor  replace into COLORG.HIST_UMQT
 NONRECOVERABLE

SET INTEGRITY FOR COLORG.HIST_UMQT ALL IMMEDIATE UNCHECKED
```

Note, the Column compression dictionaries would be built during LOAD processing and statistics would be generated

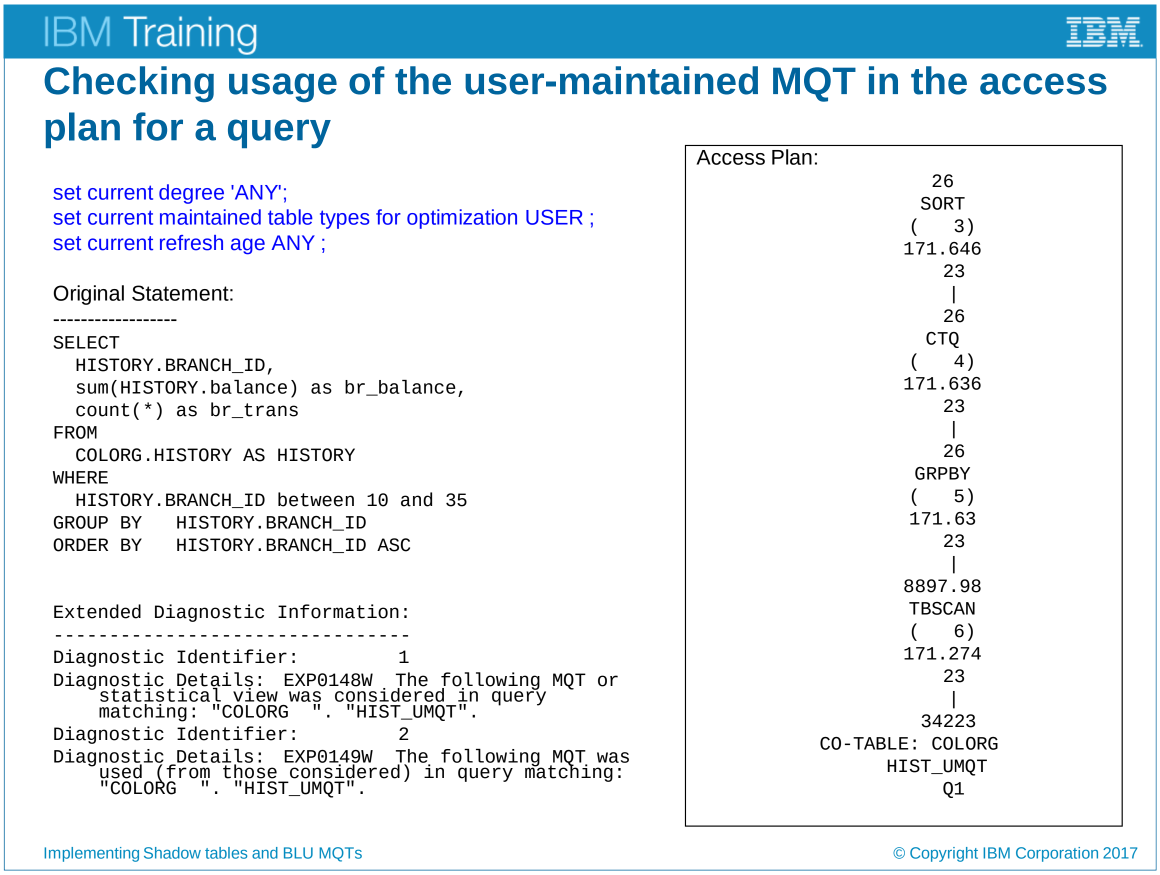
Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

Loading the data into the user-maintained MQT

The sample statements show how a column-organized user maintained MQT table could be loaded with the current data using a declared cursor with a LOAD command.

The DECLARE CURSOR statement would be similar to the SELECT statement used to create the MQT, but it could include an ORDER BY clause. You could also load data into the MQT using SQL statements.

The SET INTEGRITY statement with the IMMEDATE UNCHECKED option is used to bring the user-maintained materialized query table out of set integrity pending state.



IBM Training

Checking usage of the user-maintained MQT in the access plan for a query

```
set current degree 'ANY';
set current maintained table types for optimization USER ;
set current refresh age ANY ;

Original Statement:
------------------
SELECT
  HISTORY.BRANCH_ID,
  sum(HISTORY.balance) as br_balance,
  count(*) as br_trans
FROM
  COLORG.HISTORY AS HISTORY
WHERE
  HISTORY.BRANCH_ID between 10 and 35
GROUP BY   HISTORY.BRANCH_ID
ORDER BY   HISTORY.BRANCH_ID ASC


Extended Diagnostic Information:
--------------------------------
Diagnostic Identifier:        1
Diagnostic Details:  EXP0148W  The following MQT or
    statistical view was considered in query
    matching: "COLORG  ". "HIST_UMQT".
Diagnostic Identifier:        2
Diagnostic Details:  EXP0149W  The following MQT was
    used (from those considered) in query matching:
    "COLORG  ". "HIST_UMQT".
```

```
Access Plan:
                    26
                   SORT
                  (   3)
                  171.646
                    23
                    |
                    26
                   CTQ
                  (   4)
                  171.636
                    23
                    |
                    26
                  GRPBY
                  (   5)
                  171.63
                    23
                    |
                  8897.98
                  TBSCAN
                  (   6)
                  171.274
                    23
                    |
                  34223
           CO-TABLE: COLORG
                HIST_UMQT
                    Q1
```

Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

Checking usage of the user-maintained MQT in the access plan for a query

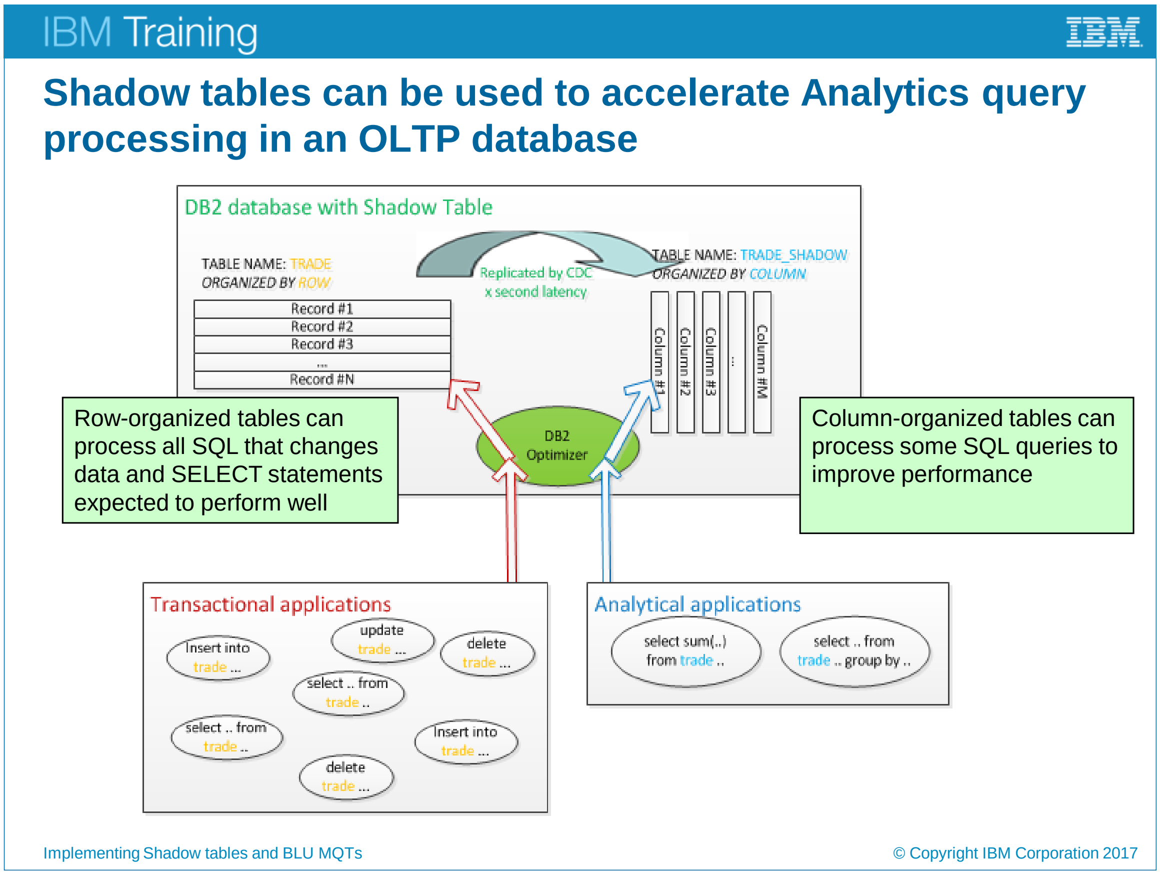
The slide shows a group of SET CURRENT statements that would be used to establish the conditions necessary to allow the user maintained MQT table to be used for a SQL query, including:

- SET CURRENT DEGREE ‘ANY’
- SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION USER
- SET CURRENT REFRESH AGE ANY

The slide includes the original SQL statement text that references the column-organized table that was used as the source for the MQT definition.

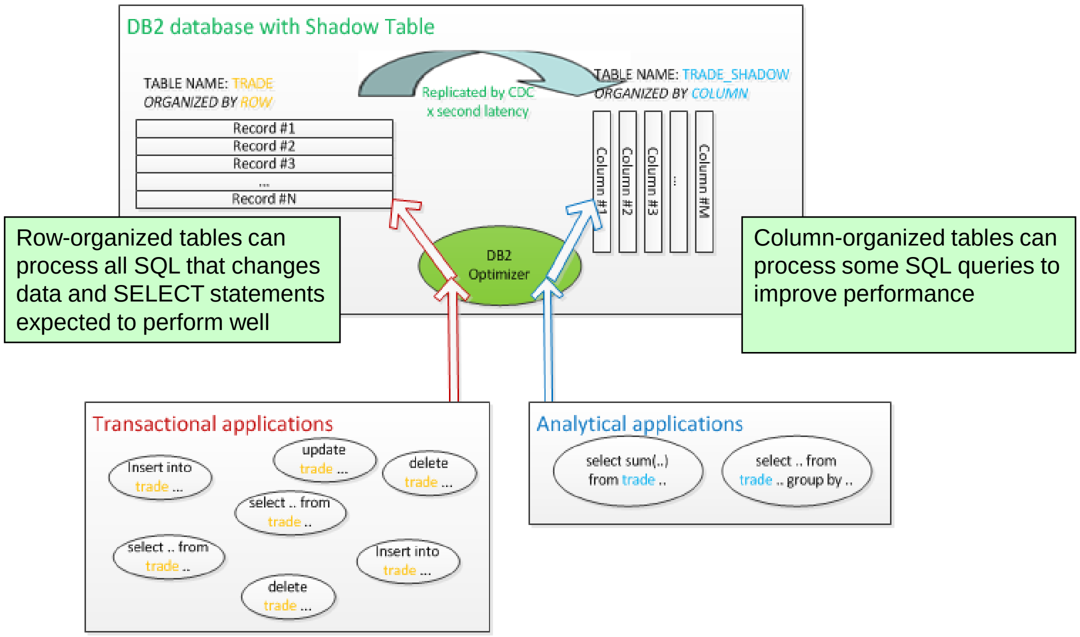
The extended diagnostic section of the DB2 **db2exfmt** explain report includes several messages stating that the optimizer found the MQT which matched the SQL statement and decided to utilize the MQT in the access plan.

The slide also includes a portion of the access plan diagram, showing the MQT table being scanned rather than the table referenced in the SQL text.



IBM Training

Shadow tables can be used to accelerate Analytics query processing in an OLTP database



Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

Shadow tables can be used to accelerate Analytics query processing in an OLTP database

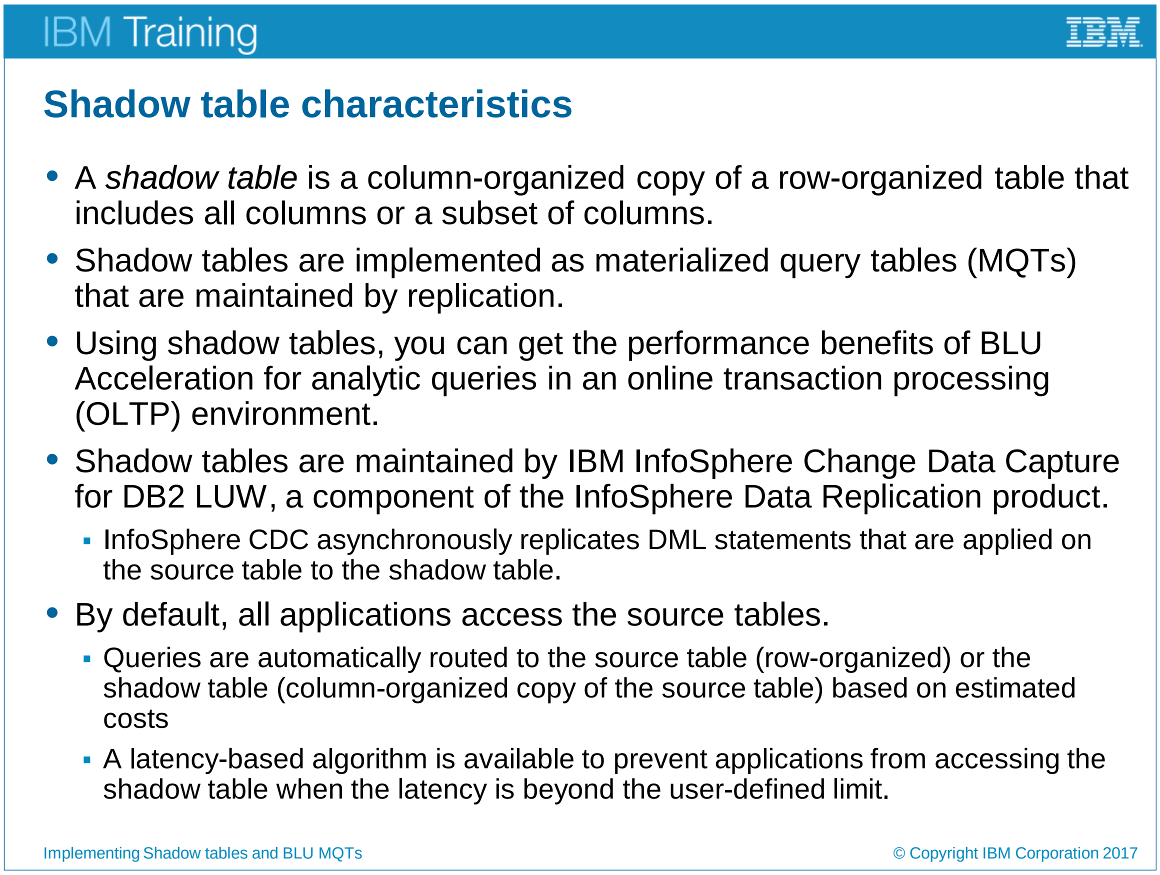
This slide shows how shadow tables can be used to accelerate analytics that are directly run against transactional data as it happens.

Within the same DB2 database, we have both the main transactional row-organized tables and their corresponding shadow tables which are copies of the source tables, but in columnar format.

With this dual format architecture, transactional applications continue to optimally target the main row-organized tables, while complex analytical queries are re-routed to the corresponding shadow tables. Since the shadow tables are in columnar format, the analytical queries are accelerated by an order of magnitude faster via BLU technology.

To maintain the shadow tables, the solution leverages Change Data Capture, an IBM InfoSphere Data Replication product. Performance testing has shown that the latency between the main transactional tables and the shadow tables can be as low as single digit seconds, allowing analytics to act on as close to real time data as possible.

Since shadow tables are used to accelerate the analytics processing, any extra indexes that were created just to speed up the analytic queries can be dropped. This may offset any impact that the data replication to shadow tables may have on the transactional workload.


IBM Training

Shadow table characteristics

- A *shadow table* is a column-organized copy of a row-organized table that includes all columns or a subset of columns.
- Shadow tables are implemented as materialized query tables (MQTs) that are maintained by replication.
- Using shadow tables, you can get the performance benefits of BLU Acceleration for analytic queries in an online transaction processing (OLTP) environment.
- Shadow tables are maintained by IBM InfoSphere Change Data Capture for DB2 LUW, a component of the InfoSphere Data Replication product.
  - InfoSphere CDC asynchronously replicates DML statements that are applied on the source table to the shadow table.
- By default, all applications access the source tables.
  - Queries are automatically routed to the source table (row-organized) or the shadow table (column-organized copy of the source table) based on estimated costs
  - A latency-based algorithm is available to prevent applications from accessing the shadow table when the latency is beyond the user-defined limit.

Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

Shadow table characteristics

A shadow table is a column-organized copy of a row-organized table that includes all columns or a subset of columns. Shadow tables are implemented as materialized query tables (MQTs) that are maintained by replication.

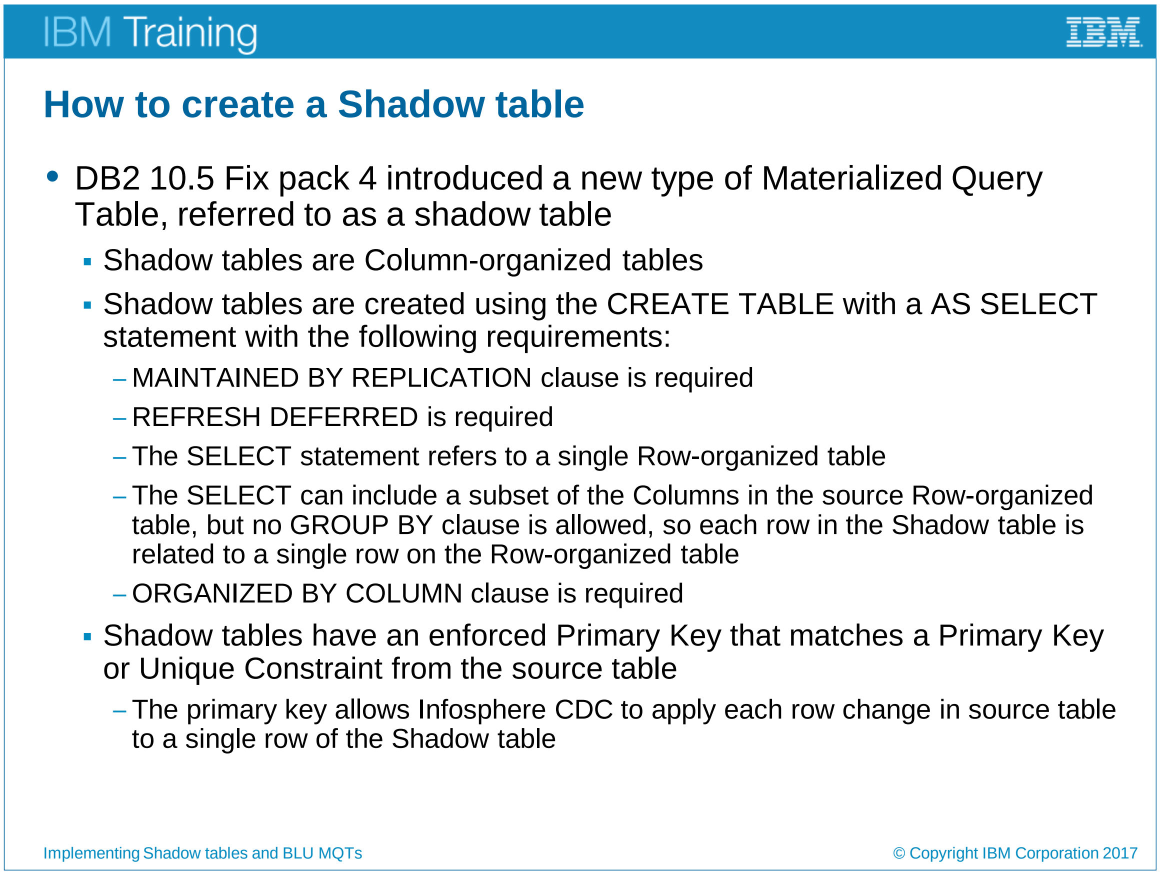
Using shadow tables, you can get the performance benefits of BLU Acceleration for analytic queries in an online transaction processing (OLTP) environment. Analytical queries against row-organized tables are automatically routed to shadow tables if the replication latency falls within a user-defined limit.

BLU Acceleration enhances the performance of complex queries through a column-organized architecture. By combining this enhanced performance for complex queries with the efficiency of row-organized tables for OLTP queries, you can use shadow tables to capitalize on the best of both worlds.

Shadow tables are maintained by IBM InfoSphere Change Data Capture for DB2, a component of the InfoSphere Data Replication product. InfoSphere CDC asynchronously replicates DML statements that are applied on the source table to the shadow table.

By default, all applications access the source tables. Queries are automatically routed to the source table (row-organized) or the shadow table (column-organized copy of the source table) by using a latency-based algorithm that prevents applications from accessing the shadow table when the latency is beyond the user-defined limit.

Shadow tables improve analytic query performance in OLTP environments without having to add indexes for this purpose.


IBM Training

How to create a Shadow table

- DB2 10.5 Fix pack 4 introduced a new type of Materialized Query Table, referred to as a shadow table
  - Shadow tables are Column-organized tables
  - Shadow tables are created using the CREATE TABLE with a AS SELECT statement with the following requirements:
    - MAINTAINED BY REPLICATION clause is required
    - REFRESH DEFERRED is required
    - The SELECT statement refers to a single Row-organized table
    - The SELECT can include a subset of the Columns in the source Row-organized table, but no GROUP BY clause is allowed, so each row in the Shadow table is related to a single row on the Row-organized table
    - ORGANIZED BY COLUMN clause is required
  - Shadow tables have an enforced Primary Key that matches a Primary Key or Unique Constraint from the source table
    - The primary key allows Infosphere CDC to apply each row change in source table to a single row of the Shadow table

Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

How to create a Shadow table

Shadow tables became available with Fix Pack 4 of DB2 10.5.

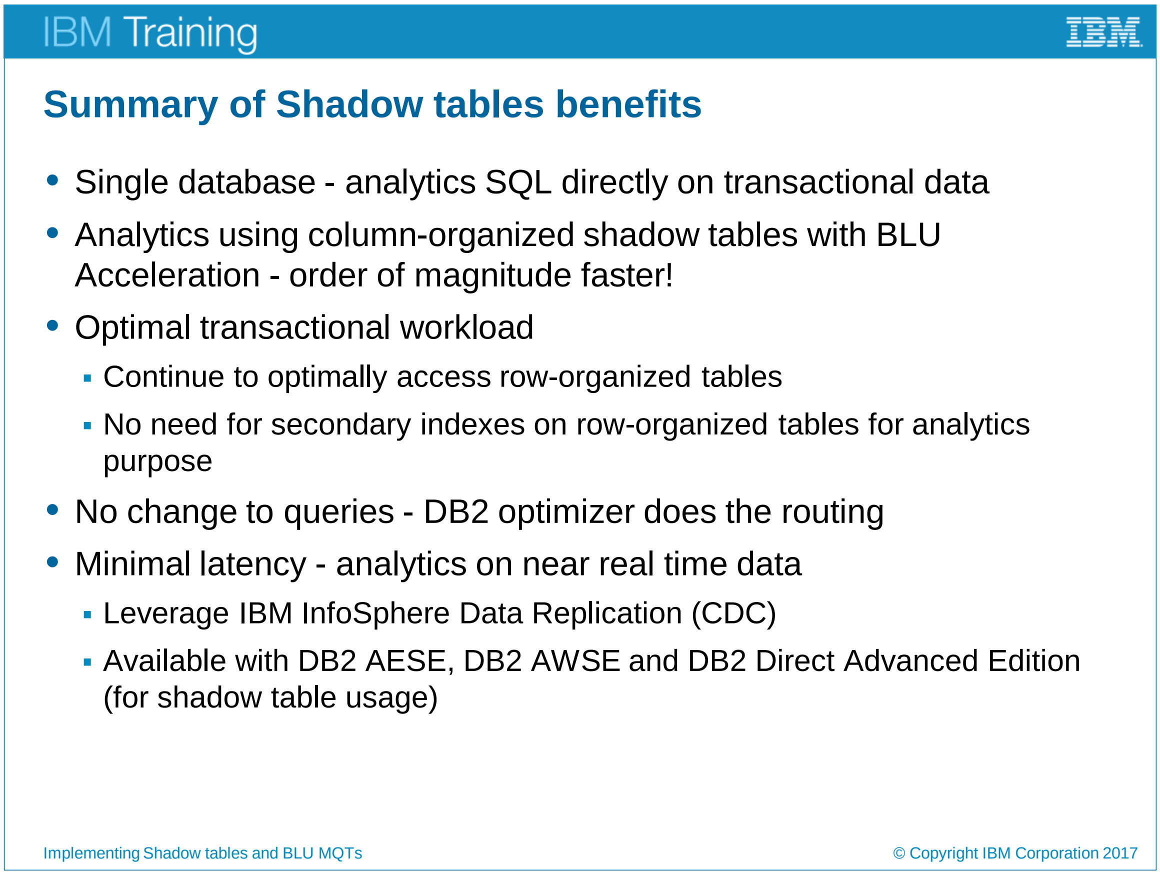
Shadow tables are a special type of Materialized Query Table (MQT).

Create the shadow table by issuing the CREATE TABLE statement with the MAINTAINED BY REPLICATION clause. This clause identifies this table as a shadow table. The primary key of the source table must be included in the select list of the CREATE TABLE statement for the shadow table.

The CREATE TABLE statement for Shadow tables must include these options:

- REFRESH DEFERRED
- ORGANIZED BY COLUMN must be specified even if the default table organization has been set to COLUMN
- The following restrictions apply to the fullselect in a shadow table definition:
  - The fullselect can reference only one base table; joins are not supported.
  - The base table must be a row-organized table.
  - The subselect can contain only a select-clause and a from-clause, no GROUP BY can be included.

- The projection list of the shadow table can reference only base table columns that are valid in a column-organized table. Expressions are not supported in the projection list. You cannot rename the columns that are referenced in the projection list by using the column list or the AS clause.
- The projection list of the shadow table must include at least one set of enforced unique constraint or primary key columns from the base table.
- The fullselect cannot include references to a nickname, a typed table, or a view or contain the SELECT DISTINCT clause.


IBM Training

Summary of Shadow tables benefits

- Single database - analytics SQL directly on transactional data
- Analytics using column-organized shadow tables with BLU Acceleration - order of magnitude faster!
- Optimal transactional workload
  - Continue to optimally access row-organized tables
  - No need for secondary indexes on row-organized tables for analytics purpose
- No change to queries - DB2 optimizer does the routing
- Minimal latency - analytics on near real time data
  - Leverage IBM InfoSphere Data Replication (CDC)
  - Available with DB2 AESE, DB2 AWSE and DB2 Direct Advanced Edition (for shadow table usage)

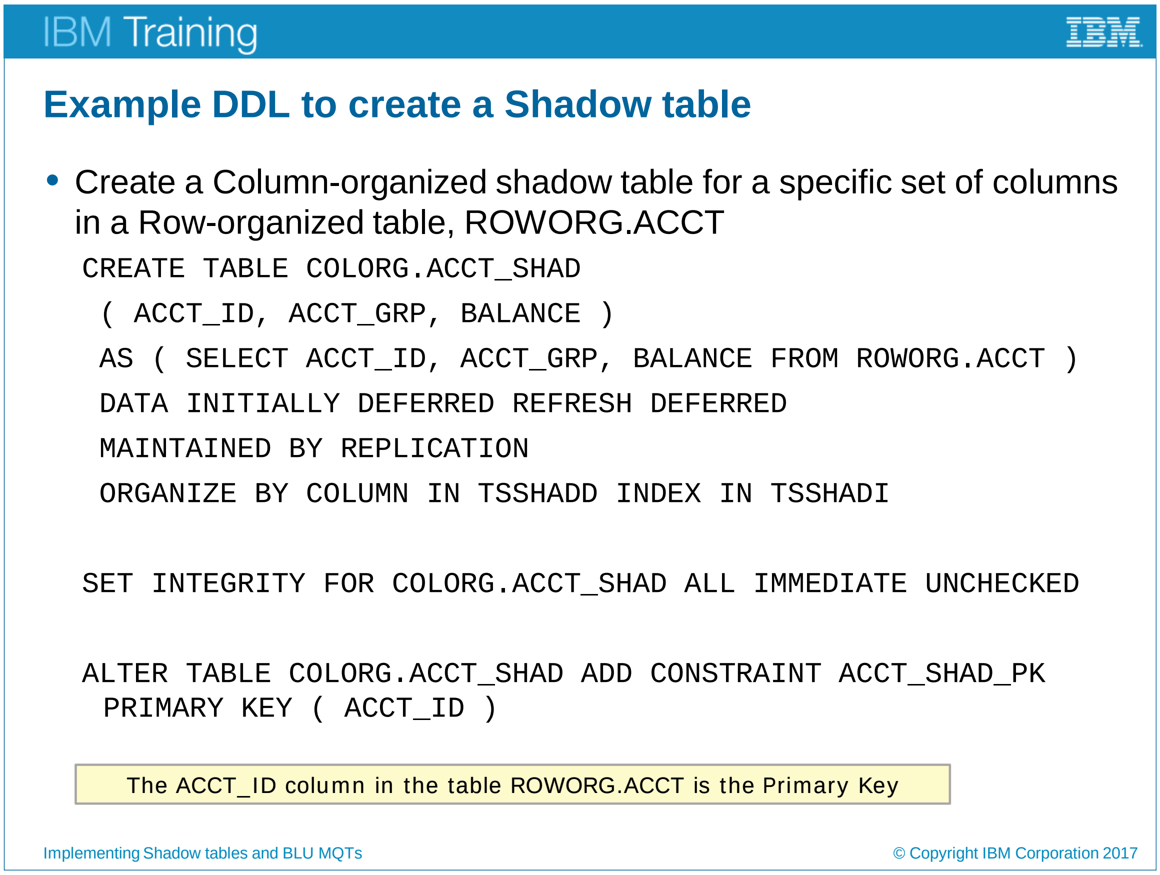
Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

Summary of Shadow table benefits

This slide summarizes the key benefits of the shadow table solution:

- A single DB2 database where analytics act directly on transactional data
- The Analytics query processing is accelerated using column-organized shadow tables that is order of magnitude faster
- The transactional workload continue to optimally access the row-organized tables.
- Using shadow tables, there is a reduced requirement for additional secondary indexes on the row-organized tables that may otherwise be needed for analytics query processing.
- No changes are required to SQL queries, since routing to shadow tables is done by the DB2 optimizer when statements are compiled. Note that only dynamic SQL can be routed to use shadow tables.

- Minimal latency is achieved by leveraging IBM InfoSphere Data Replication (CDC) which is available with the editions of DB2 LUW that support Column-organized tables:
    - DB2 AESE - DB2 Advanced Enterprise Server Edition
    - DB2 AWSE - DB2 Advanced Workgroup Server Edition
    - DB2 Direct Advanced Edition


IBM Training

Example DDL to create a Shadow table

- Create a Column-organized shadow table for a specific set of columns in a Row-organized table, ROWORG.ACCT

```
CREATE TABLE COLORG.ACCT_SHAD
 ( ACCT_ID, ACCT_GRP, BALANCE )
 AS ( SELECT ACCT_ID, ACCT_GRP, BALANCE FROM ROWORG.ACCT )
 DATA INITIALLY DEFERRED REFRESH DEFERRED
 MAINTAINED BY REPLICATION
 ORGANIZE BY COLUMN IN TSSHADD INDEX IN TSSHADI

SET INTEGRITY FOR COLORG.ACCT_SHAD ALL IMMEDIATE UNCHECKED

ALTER TABLE COLORG.ACCT_SHAD ADD CONSTRAINT ACCT_SHAD_PK
  PRIMARY KEY ( ACCT_ID )
```

The ACCT_ID column in the table ROWORG.ACCT is the Primary Key

Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

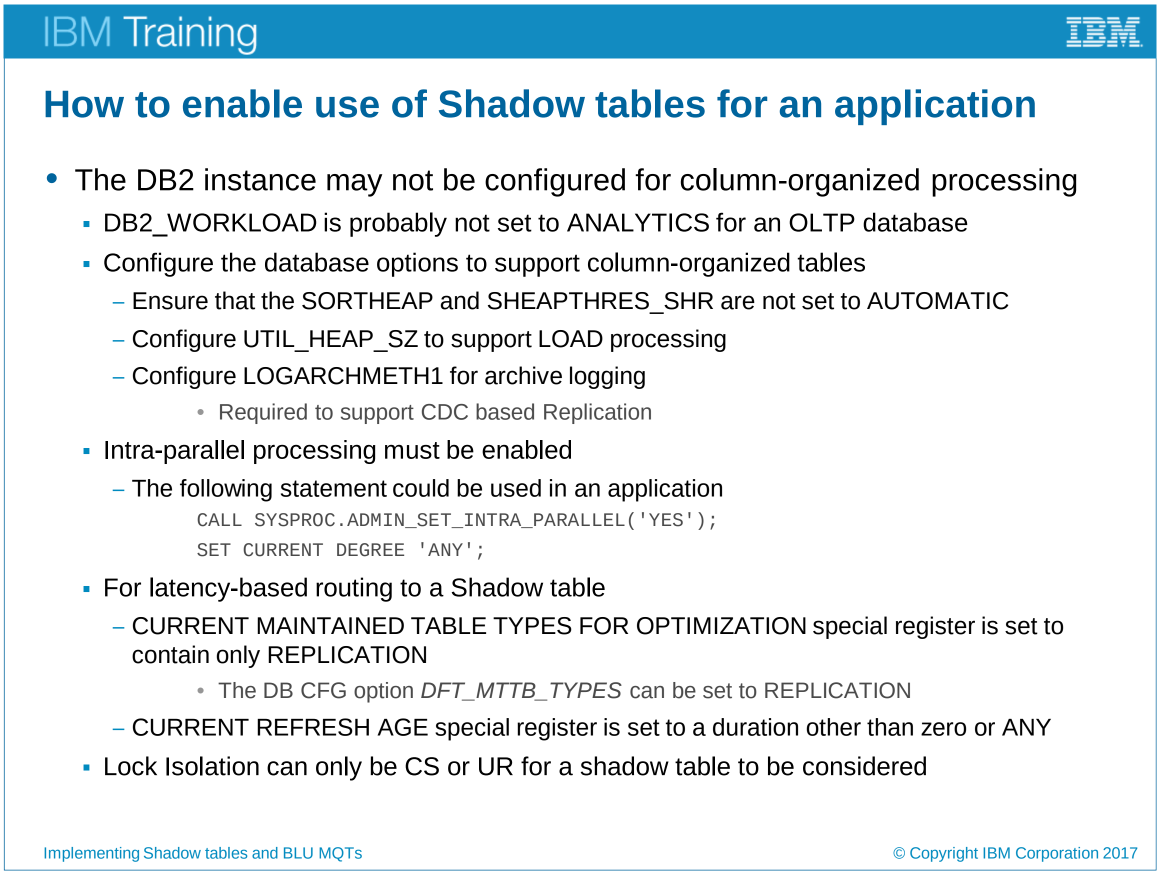
Example DDL to create a Shadow table

The slide shows an example of the statements used to create a new shadow table.

The CREATE TABLE statement refers to the row-organized table ROWORG.ACCT. The shadow table will only include the subset of columns that are expected to be referenced by analytics queries. The clauses REFRESH DEFERRED, MAINTAINED BY REPLICATION and ORGANIZED BY COLUMN are included.

The SET INTEGRITY statement with the IMMEDIATE UNCHECKED option resolves the set integrity pended status for the new shadow table.

The ALTER TABLE statement defines an enforced primary key for the shadow table based on the primary key column for the row-organized table.



IBM Training

How to enable use of Shadow tables for an application

- The DB2 instance may not be configured for column-organized processing
  - DB2_WORKLOAD is probably not set to ANALYTICS for an OLTP database
  - Configure the database options to support column-organized tables
    - Ensure that the SORTHEAP and SHEAPTHRES_SHR are not set to AUTOMATIC
    - Configure UTIL_HEAP_SZ to support LOAD processing
    - Configure LOGARCHMETH1 for archive logging
      - Required to support CDC based Replication
  - Intra-parallel processing must be enabled
    - The following statement could be used in an application

```
CALL SYSPROC.ADMIN_SET_INTRA_PARALLEL('YES');
SET CURRENT DEGREE 'ANY';
```

  - For latency-based routing to a Shadow table
    - CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register is set to contain only REPLICATION
      - The DB CFG option *DFT_MTTB_TYPES* can be set to REPLICATION
    - CURRENT REFRESH AGE special register is set to a duration other than zero or ANY
  - Lock Isolation can only be CS or UR for a shadow table to be considered

Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

How to enable use of Shadow tables for an application

It is quite possible that Shadow tables would be defined in an existing DB2 database where the OLTP processing is currently performed.

That DB2 instance and database would probably not been configured using the DB2_WORKLOAD registry variable setting of ANALYTICS.

Since shadow tables are column-organized tables, the DB2 database will need to meet the basic requirements for BLU Acceleration processing, including:

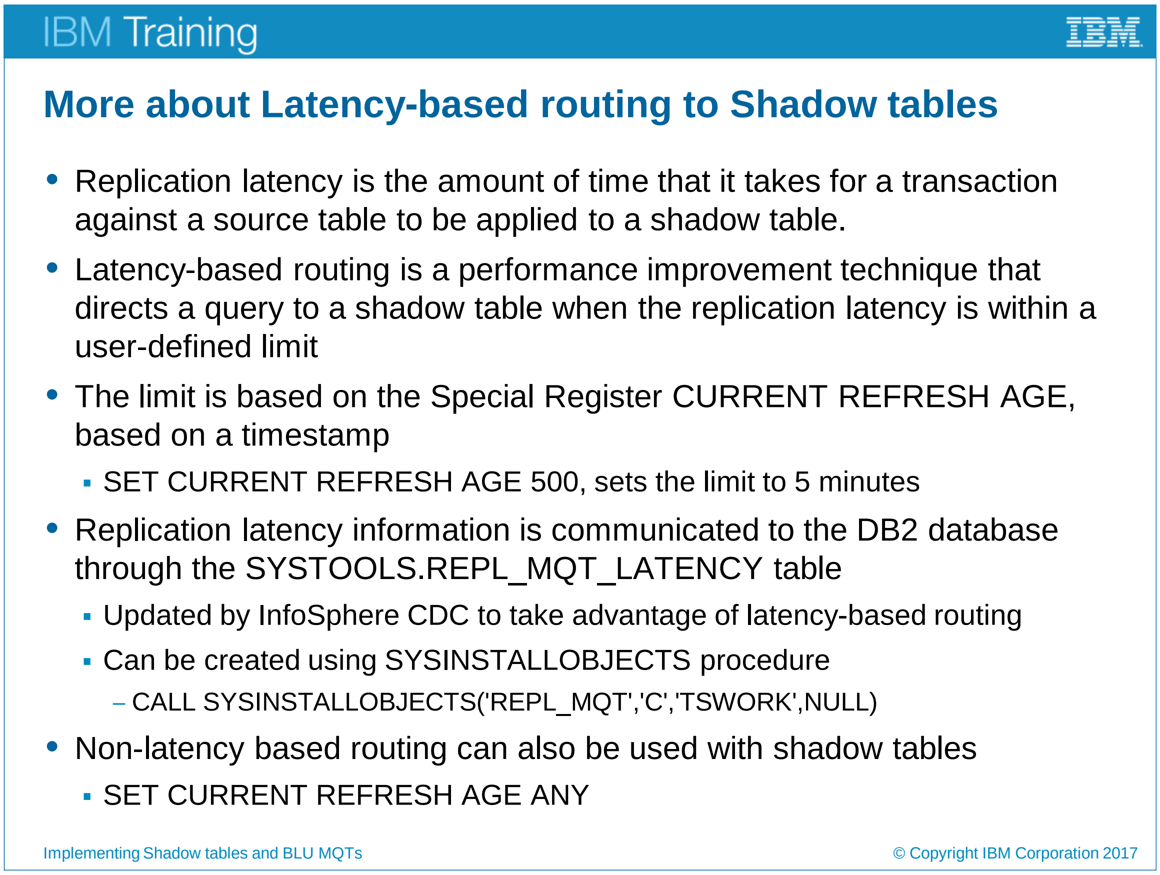
- The database configuration options SORTHEAP and SHEAPTHRES_SHR cannot be set to AUTOMATIC, but need to be large enough to process column-organized tables.
- The utility heap configuration option, UTIL_HEAP_SZ, needs to be large enough to support efficient LOAD processing for column-organized tables.
- The database needs to be configured for archive logging, using LOGARCHMETH1, since the Infosphere Change Data Capture software needs to be able to access archived logs.

Column-organized processing in DB2 requires intra-parallel processing being enabled. One method to enable intra-parallel processing is for the application to call the procedure ADMIN_SET_INTRA_PARALLEL and to set the special register CURRENT DEGREE to ANY.

Latency-based routing is a performance improvement technique that directs a query to a shadow table when the replication latency is within a user-defined limit.

If you create a shadow table with ENABLE QUERY OPTIMIZATION clause, each of the following conditions must be true to optimize query processing that is based on a latency period:

- The CURRENT QUERY OPTIMIZATION special register is set to 2 or a value greater than or equal to 5.
- The CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register is set to contain **only** REPLICATION.
- The CURRENT REFRESH AGE special register is set to a duration other than zero or ANY.



IBM Training

More about Latency-based routing to Shadow tables

- Replication latency is the amount of time that it takes for a transaction against a source table to be applied to a shadow table.
- Latency-based routing is a performance improvement technique that directs a query to a shadow table when the replication latency is within a user-defined limit
- The limit is based on the Special Register CURRENT REFRESH AGE, based on a timestamp
  - SET CURRENT REFRESH AGE 500, sets the limit to 5 minutes
- Replication latency information is communicated to the DB2 database through the SYSTOOLS.REPL_MQT_LATENCY table
  - Updated by InfoSphere CDC to take advantage of latency-based routing
  - Can be created using SYSINSTALLOBJECTS procedure
    - CALL SYSINSTALLOBJECTS('REPL_MQT','C','TSWORK',NULL)
- Non-latency based routing can also be used with shadow tables
  - SET CURRENT REFRESH AGE ANY

Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

More about Latency-based routing to Shadow Tables

Replication latency is the amount of time that it takes for a transaction against a source table to be applied to a shadow table. Latency-based routing is a performance improvement technique that directs a query to a shadow table when the replication latency is within a user-defined limit.

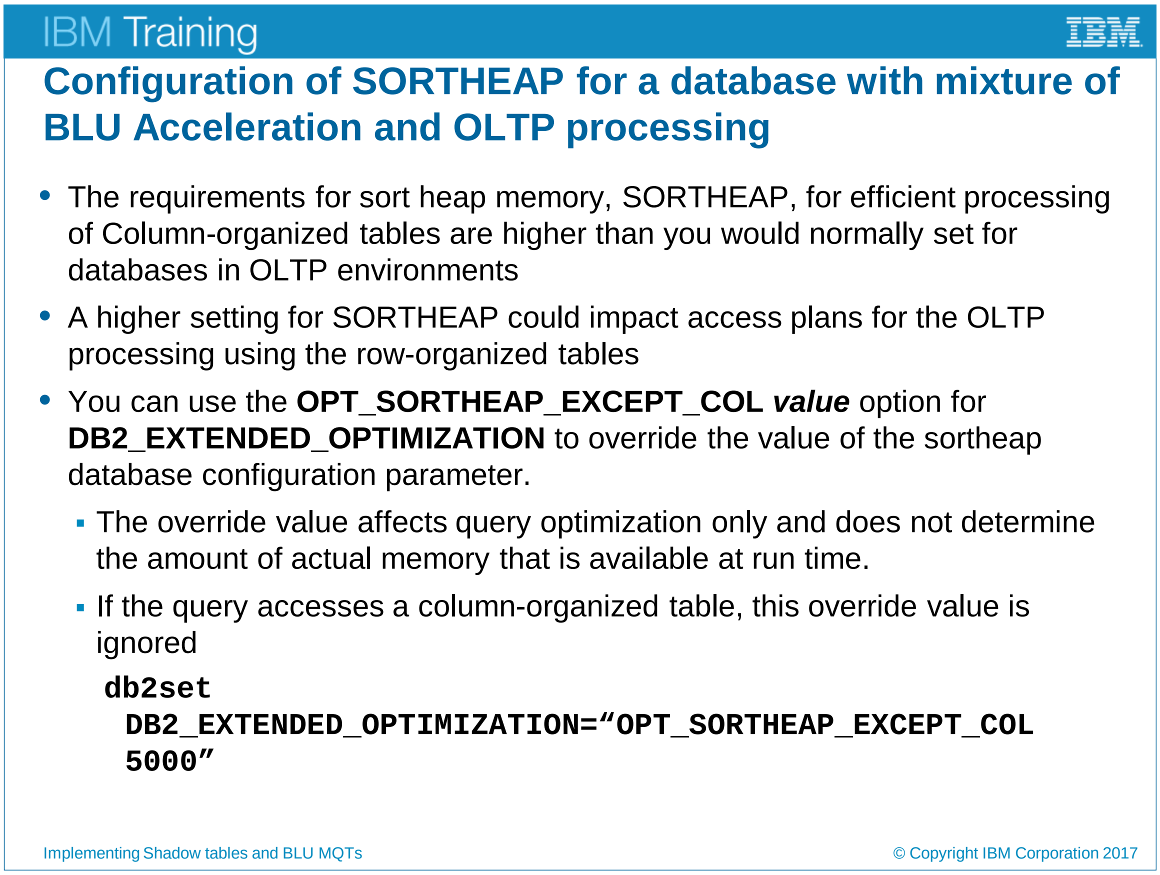
If you create a shadow table with ENABLE QUERY OPTIMIZATION clause, each of the following conditions must be true to optimize query processing that is based on a latency period:

- The CURRENT QUERY OPTIMIZATION special register is set to 2 or a value greater than or equal to 5.
- The CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register is set to contain only REPLICATION.
- The CURRENT REFRESH AGE special register is set to a duration other than zero or ANY. This special register specifies the refresh age as a timestamp with a format of yyyymmddhhmmss where the parts of the timestamp are defined as follows:
  - yyyy indicates the year (0-9999)
  - mm indicates the month (0-11)

- dd indicates the day (0-30)
- hh indicates the hour (0-23)
- mm indicates the minute (0-5)
- ss indicates the second (0-59)
- The timestamp value can be truncated on the left. This left truncation means that you do not have to specify year, month, day, and so on, if you want a refresh age of only one second. However, individual elements that are preceded by another element must include any leading zeros. For example, a refresh age of 10705 represents 1 hour, 7 minutes, and 5 seconds.

Replication latency information is communicated to the DB2 instance through the SYSTOOLS.REPL_MQT_LATENCY table that is updated by InfoSphere CDC to take advantage of latency-based routing. This table can be created using the DB2 stored procedure SYSINSTALLOBJECTS.

Shadow tables could also be used without latency checking by setting the CURRENT REFRESH AGE special register to ANY.



IBM Training

Configuration of SORTHEAP for a database with mixture of BLU Acceleration and OLTP processing

- The requirements for sort heap memory, SORTHEAP, for efficient processing of Column-organized tables are higher than you would normally set for databases in OLTP environments
- A higher setting for SORTHEAP could impact access plans for the OLTP processing using the row-organized tables
- You can use the **OPT_SORTHEAP_EXCEPT_COL *value*** option for **DB2_EXTENDED_OPTIMIZATION** to override the value of the sortheap database configuration parameter.
  - The override value affects query optimization only and does not determine the amount of actual memory that is available at run time.
  - If the query accesses a column-organized table, this override value is ignored

```
db2set
  DB2_EXTENDED_OPTIMIZATION=“OPT_SORTHEAP_EXCEPT_COL
  5000”
```

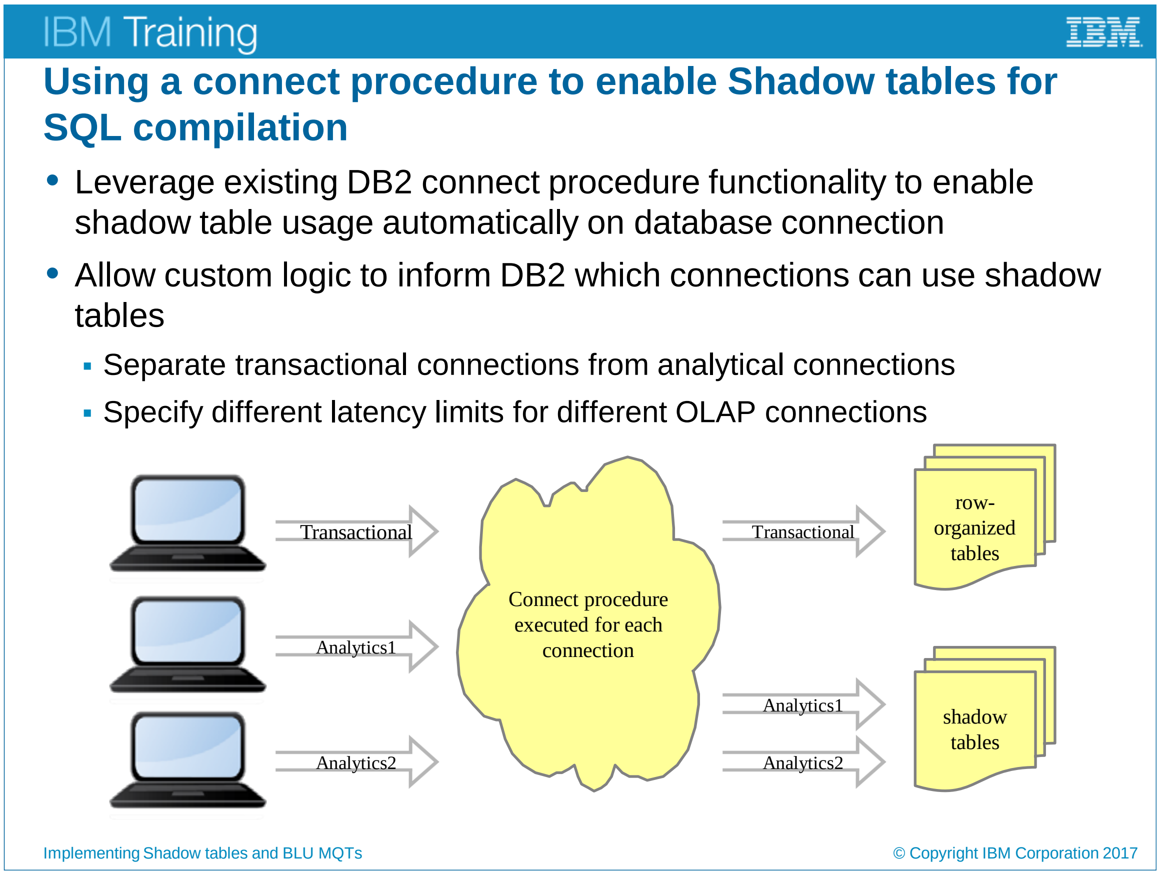
Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

Configuration of SORTHEAP for a database with mixture of BLU Acceleration and OLTP processing

You can use the OPT_SORTHEAP_EXCEPT_COL value option of the DB2 registry variable DB2_EXTENDED_OPTIMIZATION to override the value of the sortheap database configuration parameter.

The override value affects query optimization only and does not determine the amount of actual memory that is available at run time. If the query accesses a column-organized table, this override value is ignored to allow the query compiler to use the current value of the sortheap database configuration parameter.

One usage of the OPT_SORTHEAP_EXCEPT_COL is for shadow tables. Shadow tables facilitate BLU Acceleration for analytical queries in OLTP environment. Shadow tables are column-organized tables. The requirements for sort heap memory are higher than you would normally have for databases in OLTP environments. To increase the sort heap memory without affecting existing access plans for OLTP queries, add OPT_SORTHEAP_EXCEPT_COL to DB2_EXTENDED_OPTIMIZATION to override the value of the sortheap database configuration parameter.



IBM Training

Using a connect procedure to enable Shadow tables for SQL compilation

- Leverage existing DB2 connect procedure functionality to enable shadow table usage automatically on database connection
- Allow custom logic to inform DB2 which connections can use shadow tables
  - Separate transactional connections from analytical connections
  - Specify different latency limits for different OLAP connections

Implementing Shadow tables and BLU MQTs

© Copyright IBM Corporation 2017

Using a connect procedure to enable Shadow tables for SQL compilation

A connect procedure can be used to automatically enable usage of shadow tables for selected applications without the need of modifying the applications.

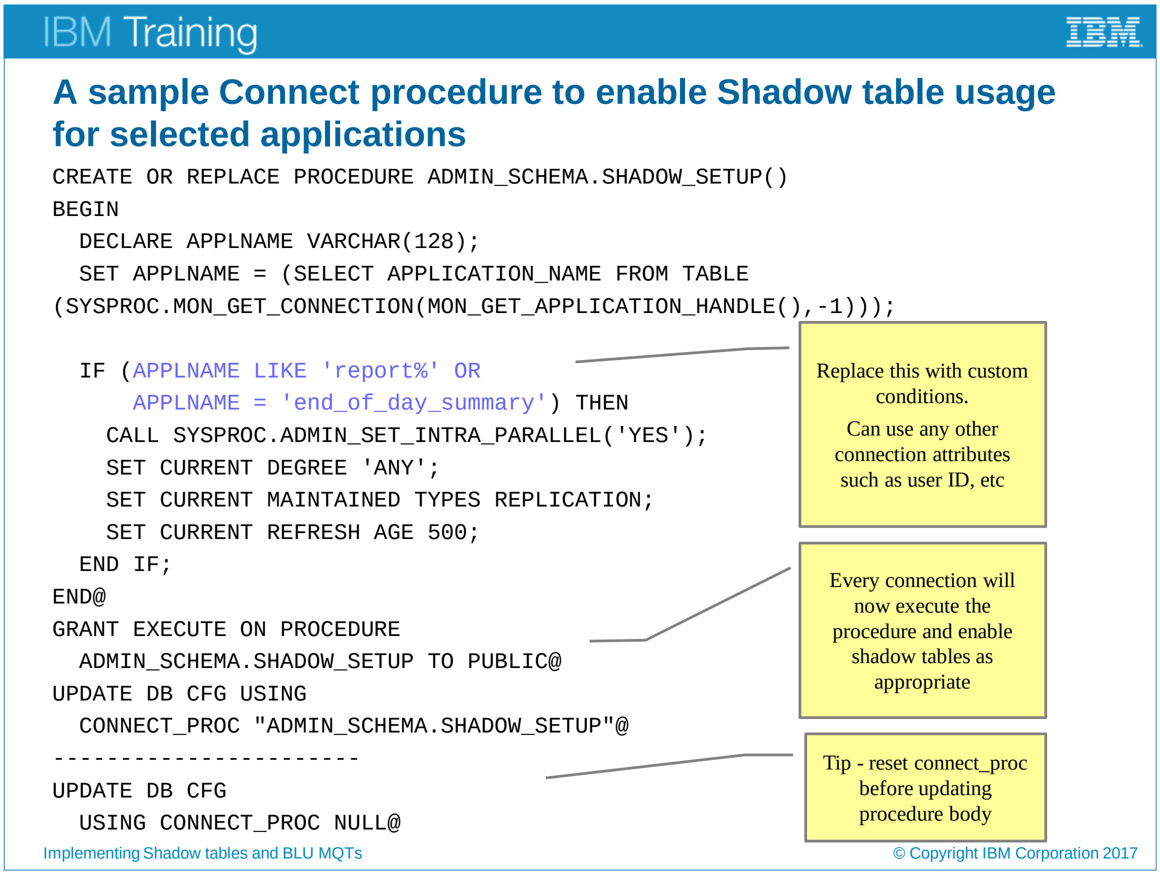
Basically, a connect procedure is a SQL procedure that is executed for each connection to the database.

The body of the procedure can be implemented to test attributes of each connection and conditionally execute those SQL statements on the previous slide only for those connections corresponding to applications that have been identified to be able to make use of shadow tables.

The diagram on this visual shows three connections:

- One transactional connection that should not make use of shadow tables,
- Two analytical connections that have been identified to make use of shadow tables.

The connect procedure is executed for all three connections but only those analytical connections are allowed to make use of shadow tables while the transactional one continues to only access the row-organized tables.



IBM Training

A sample Connect procedure to enable Shadow table usage for selected applications

```
CREATE OR REPLACE PROCEDURE ADMIN_SCHEMA.SHADOW_SETUP()
BEGIN
  DECLARE APPLNAME VARCHAR(128);
  SET APPLNAME = (SELECT APPLICATION_NAME FROM TABLE
(SYSPROC.MON_GET_CONNECTION(MON_GET_APPLICATION_HANDLE(),-1)));

  IF (APPLNAME LIKE 'report%' OR
      APPLNAME = 'end_of_day_summary') THEN
    CALL SYSPROC.ADMIN_SET_INTRA_PARALLEL('YES');
    SET CURRENT DEGREE 'ANY';
    SET CURRENT MAINTAINED TYPES REPLICATION;
    SET CURRENT REFRESH AGE 500;
  END IF;
END@
GRANT EXECUTE ON PROCEDURE
  ADMIN_SCHEMA.SHADOW_SETUP TO PUBLIC@
UPDATE DB CFG USING
  CONNECT_PROC "ADMIN_SCHEMA.SHADOW_SETUP"@
-----------------------
UPDATE DB CFG
  USING CONNECT_PROC NULL@
```

Replace this with custom conditions.
Can use any other connection attributes such as user ID, etc

Every connection will now execute the procedure and enable shadow tables as appropriate

Tip - reset connect_proc before updating procedure body

Implementing Shadow tables and BLU MQTs

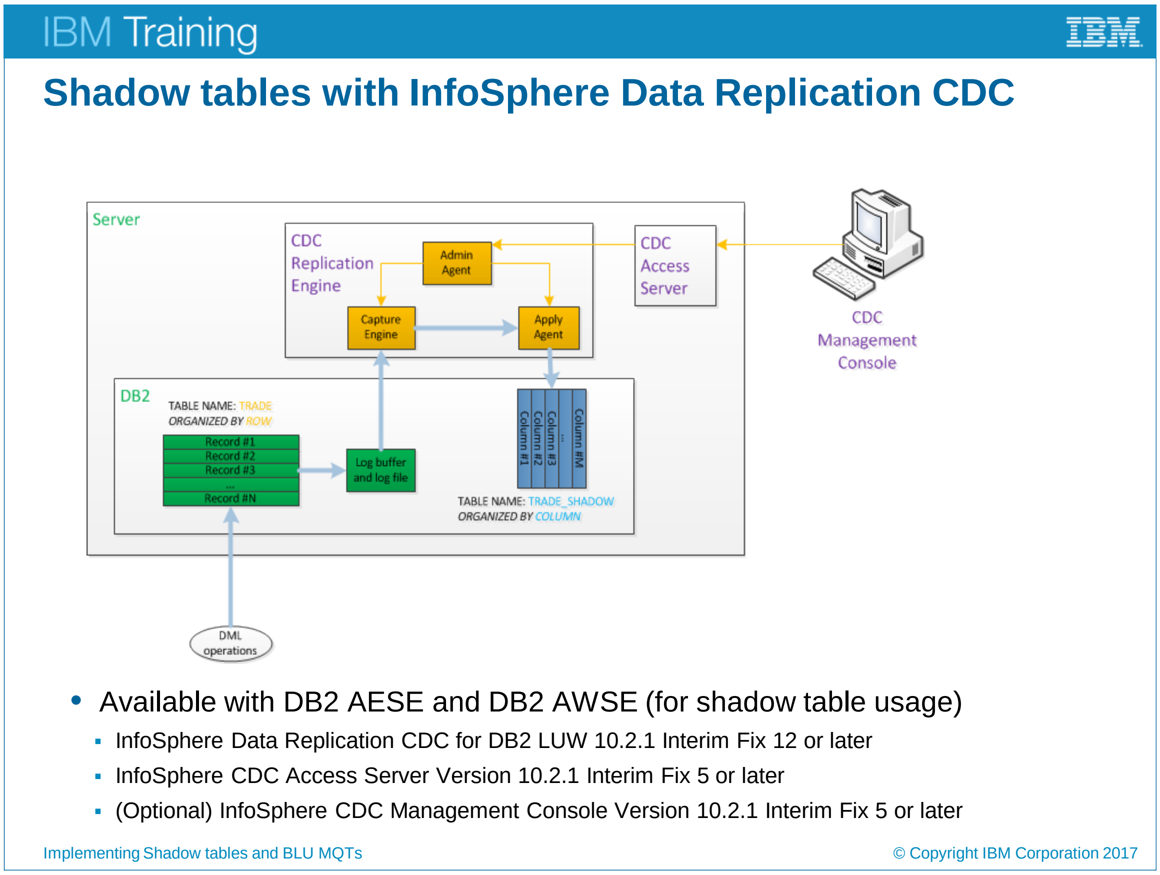
© Copyright IBM Corporation 2017

A sample Connect procedure to enable Shadow table usage for selected applications

Here we have a sample connect procedure.

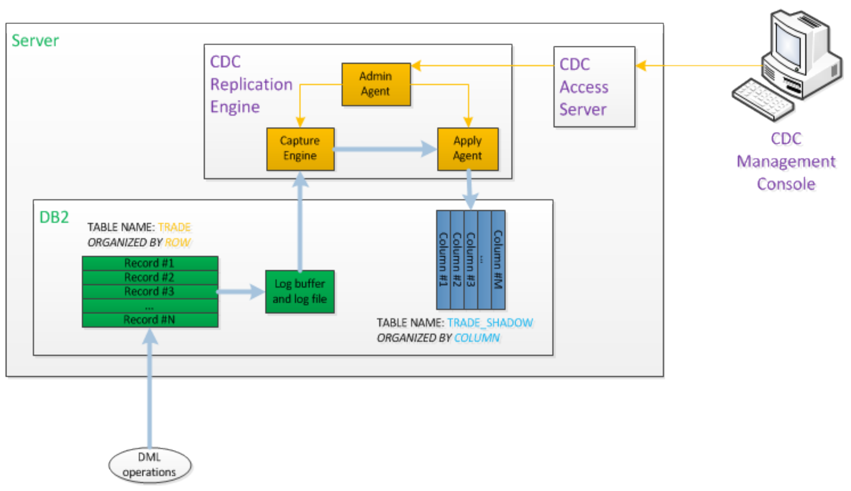
The body is quite simple. It tests the application name of the connection to look for certain analytic applications that have been identified to be able to make use of shadow tables. In this sample, only for these analytic applications will DB2 consider routing to shadow tables, all other applications are not impacted by the presence of shadow tables.

Since a connect procedure is just a standard SQL procedure, you can customize it to fit your needs to allow you to properly and effectively identify which connections should make use of shadow tables. For example, instead of hard-coding connection attributes in the procedure, you can setup an “opt-in” DB2 table containing applications that can make use of shadow tables and have the connect procedure search this “opt-in” DB2 table to determine the connections for which to enable usage of shadow tables.



IBM Training

Shadow tables with InfoSphere Data Replication CDC



- Available with DB2 AESE and DB2 AWSE (for shadow table usage)
  - InfoSphere Data Replication CDC for DB2 LUW 10.2.1 Interim Fix 12 or later
  - InfoSphere CDC Access Server Version 10.2.1 Interim Fix 5 or later
  - (Optional) InfoSphere CDC Management Console Version 10.2.1 Interim Fix 5 or later

Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

Shadow tables with InfoSphere Data Replication CDC

This is a pictorial view illustrating how DB2 and CDC work together to maintain the shadow tables.

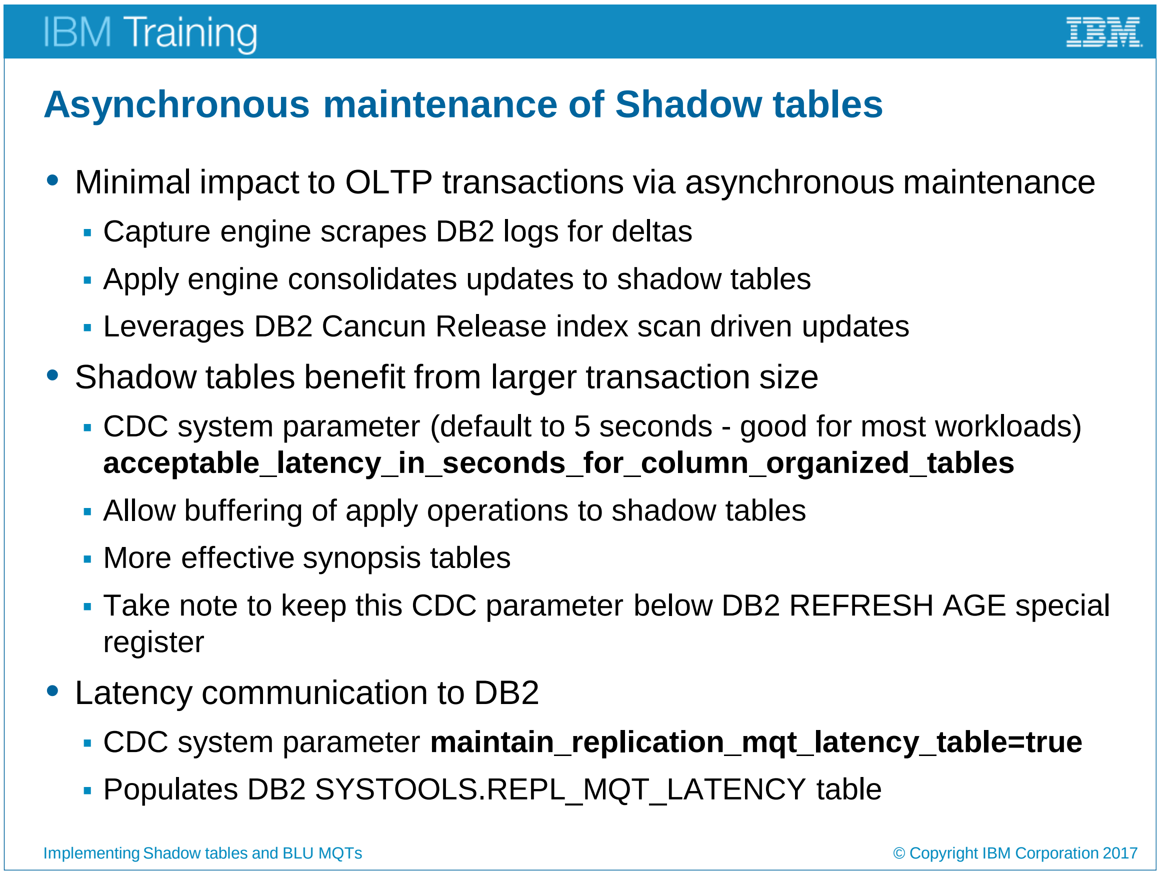
Infosphere CDC is composed of various components

- The CDC Replication Engine is the main component that reads from the DB2 logs and apply the corresponding updates to the shadow tables.
- The CDC Access Server is the component that manages various aspect of data replication such as setting up data stores, replication subscriptions and so on.
- Both the Replication Engine and the Access Server are required for data replication and these components can be installed on the same server as DB2.
- The CDC Management Console is the graphical interface to Access Server. This is an optional component that can be used to manage the data replication from a desktop/laptop.

The slide shows that the row-organized tables and column-organized shadow tables are defined in the same DB2 database. The changes logged for the row-organized tables are read by the Capture agent of CDC and then used by the Apply agent of CDC to update the column-organized shadow tables. The CDC software is running on the same server as the DB2 database.

To support shadow tables, you must install a supported version of the following InfoSphere CDC software components:

- InfoSphere CDC for DB2 for LUW Version 10.2.1 Interim Fix 12 or later releases.
- InfoSphere CDC Access Server Version 10.2.1 Interim Fix 5 or later releases.
- InfoSphere CDC Management Console Version 10.2.1 Interim Fix 5 or later



IBM Training

Asynchronous maintenance of Shadow tables

- Minimal impact to OLTP transactions via asynchronous maintenance
  - Capture engine scrapes DB2 logs for deltas
  - Apply engine consolidates updates to shadow tables
  - Leverages DB2 Cancun Release index scan driven updates
- Shadow tables benefit from larger transaction size
  - CDC system parameter (default to 5 seconds - good for most workloads) **acceptable_latency_in_seconds_for_column_organized_tables**
  - Allow buffering of apply operations to shadow tables
  - More effective synopsis tables
  - Take note to keep this CDC parameter below DB2 REFRESH AGE special register
- Latency communication to DB2
  - CDC system parameter **maintain_replication_mqt_latency_table=true**
  - Populates DB2 SYSTOOLS.REPL_MQT_LATENCY table

Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

Asynchronous maintenance of Shadow tables

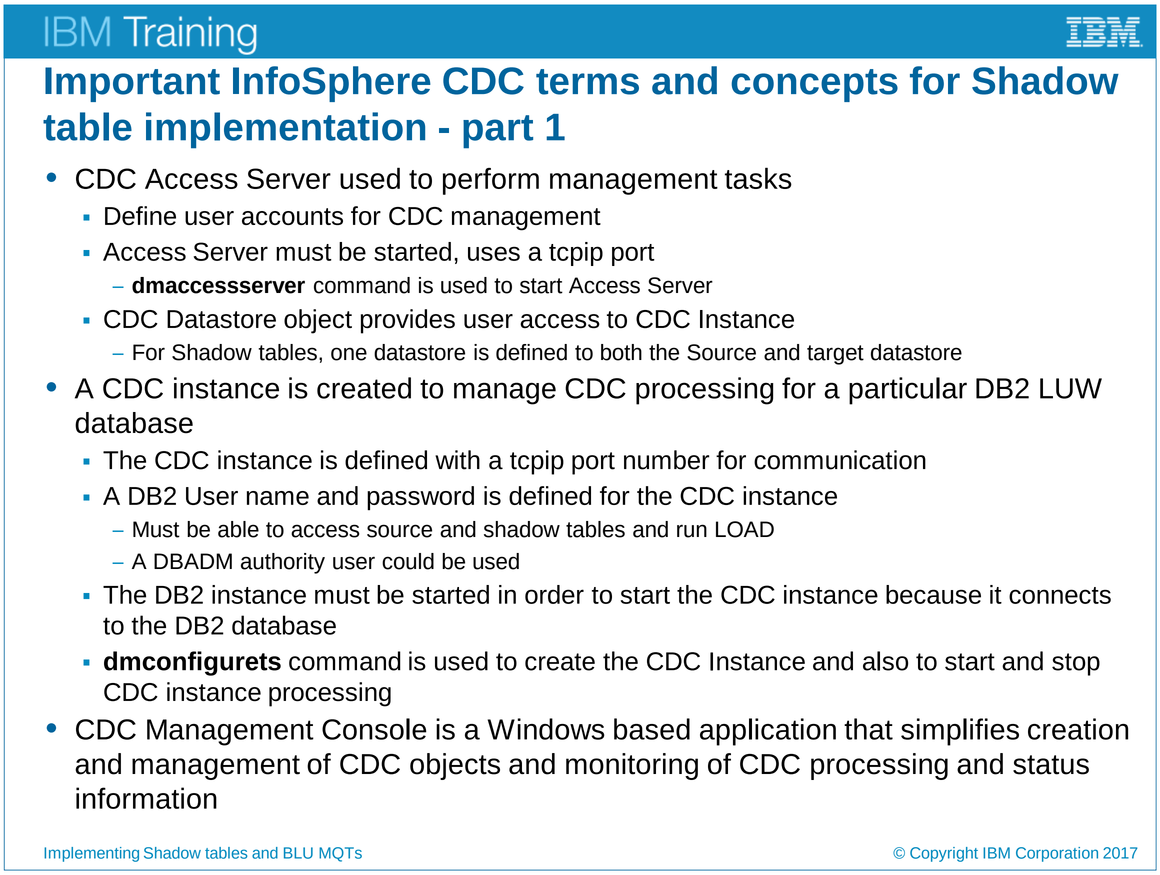
CDC maintenance of the shadow tables is an asynchronous process and hence has minimal impact to OLTP transactions. As mentioned previously, with shadow tables, you may decide to drop any existing extra indexes that were created to speed up analytic queries which may offset any overhead that the CDC replication may have on the OLTP transactions.

CDC replication involves a capture engine that scrapes the DB2 logs for update deltas and feed these deltas to the apply engine to maintain the shadow tables. The apply to the shadow tables is further optimized to leverage the enhanced index scan driven updates for column-organized tables with DB2 10.5 fix pack 4.

Normally, CDC will apply deltas to the target tables as quickly as it can. However, for shadow tables and column organized tables in general, it is advantageous to apply deltas to the target in chunks to increases the size of each apply transaction. CDC offers a system parameter, **acceptable_latency_in_seconds_for_column_organized_tables** to accomplish this by delaying the ‘apply’ to the target tables, up to a certain number of seconds. The default is 5 seconds, which should be good enough for most workloads.

Note that this delay will affect the replication latency and hence should be set to a value that is under the latency limit that your analytic applications can tolerate as specified by the REFRESH AGE special register, else DB2 may not route the analytic queries to the shadow tables.

As mentioned previously, CDC communicates the latency info to DB2 via the SYSTOOLS.REPL_MQT_LATENCY table. A CDC system parameter, **maintain_replication_mqt_latency_table** needs to be set to TRUE for CDC to populate this table with the latency info.


IBM Training

Important InfoSphere CDC terms and concepts for Shadow table implementation - part 1

- CDC Access Server used to perform management tasks
  - Define user accounts for CDC management
  - Access Server must be started, uses a tcpip port
    - **dmaccessserver** command is used to start Access Server
  - CDC Datastore object provides user access to CDC Instance
    - For Shadow tables, one datastore is defined to both the Source and target datastore
- A CDC instance is created to manage CDC processing for a particular DB2 LUW database
  - The CDC instance is defined with a tcpip port number for communication
  - A DB2 User name and password is defined for the CDC instance
    - Must be able to access source and shadow tables and run LOAD
    - A DBADM authority user could be used
  - The DB2 instance must be started in order to start the CDC instance because it connects to the DB2 database
  - **dmconfigurets** command is used to create the CDC Instance and also to start and stop CDC instance processing
- CDC Management Console is a Windows based application that simplifies creation and management of CDC objects and monitoring of CDC processing and status information

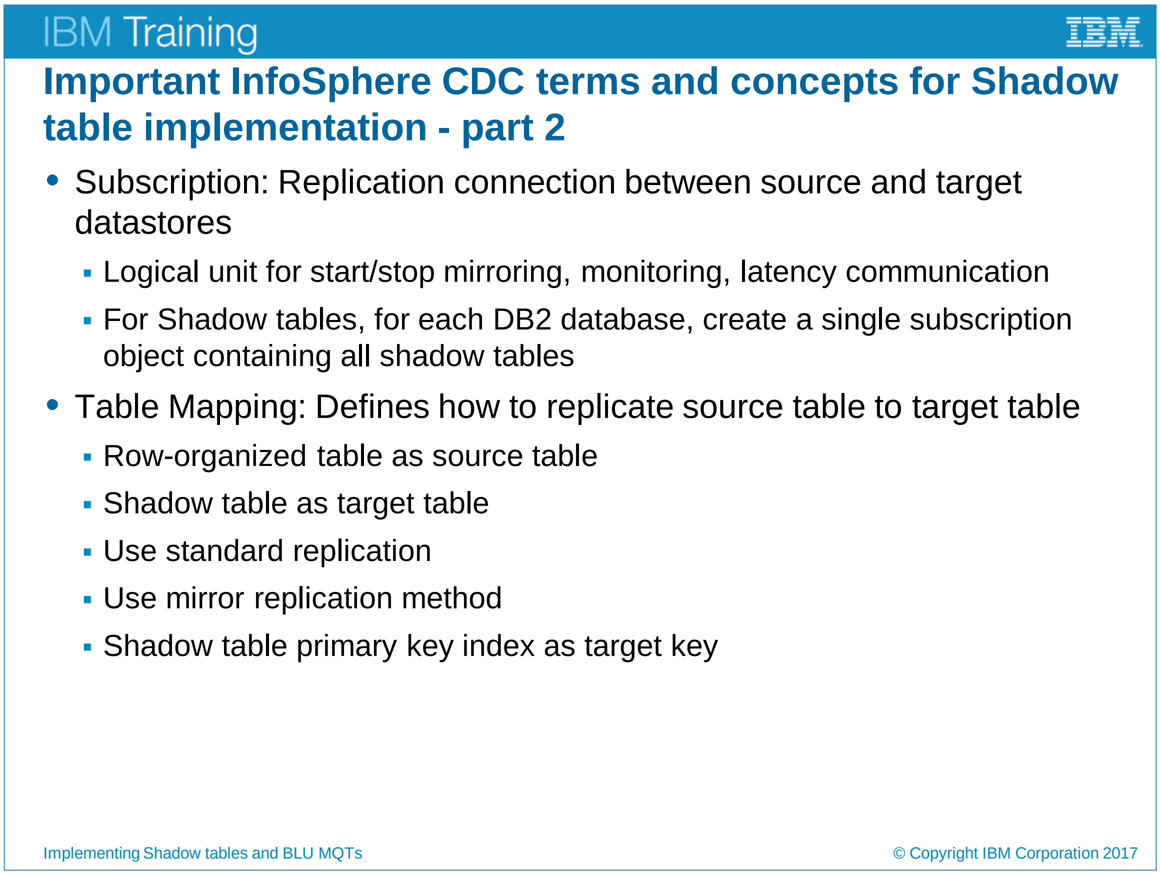
Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

Important InfoSphere CDC terms and concepts for Shadow table implementation - part 1

Shadow tables require the following InfoSphere CDC software components:

- **InfoSphere CDC for DB2 for LUW**
  - This software is the replication engine for DB2 for Linux, UNIX, and Windows.
  - The replication engine reads the transaction logs and captures all the DML operations for the source row-organized table. Then, the apply agent applies these changes to the shadow table.
  - A CDC instance will be created for each DB2 database that will use CDC to replicate the changes to shadow tables.
  - The command **dmconfigurets** can be used to create, edit, start and stop a CDC instance.
  - A DB2 user name is defined to establish a connection to the DB2 database associated with the CDC instance. In order to start the CDC instance the DB2 instance will need to be started.

- **InfoSphere CDC Access Server**
    - This software is a server application that directs communications between the replication engine processes and the InfoSphere CDC Management Console command line processor (CHCCLP).
    - You will use access server to define CDC user profiles.
    - Access server is also used to create a CDC object called a datastore that is used by a CDC user to work with a CDC Instance.
- **InfoSphere CDC Management Console**
    - This software is an administration application that you can use to configure and to monitor replication for shadow tables. This GUI interface runs on only Windows operating systems. It includes an event log and a monitoring tool.
    - A datastore is an abstraction that represents an InfoSphere CDC instance. It holds information about the database and data files that are required for replication.
    - InfoSphere CDC Management Console and the CHCCLP command-line interface interact with the database by connecting to only a datastore. While general InfoSphere CDC environments contain source and target datastores, shadow tables require only one datastore because the source and target are the same database.


IBM Training

Important InfoSphere CDC terms and concepts for Shadow table implementation - part 2

- Subscription: Replication connection between source and target datastores
  - Logical unit for start/stop mirroring, monitoring, latency communication
  - For Shadow tables, for each DB2 database, create a single subscription object containing all shadow tables
- Table Mapping: Defines how to replicate source table to target table
  - Row-organized table as source table
  - Shadow table as target table
  - Use standard replication
  - Use mirror replication method
  - Shadow table primary key index as target key

Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

Important InfoSphere CDC terms and concepts for Shadow table implementation - part 2

Subscriptions

A subscription is a container for table mappings. It logically links source and target datastores and contains various parameters that determine the replication behavior.

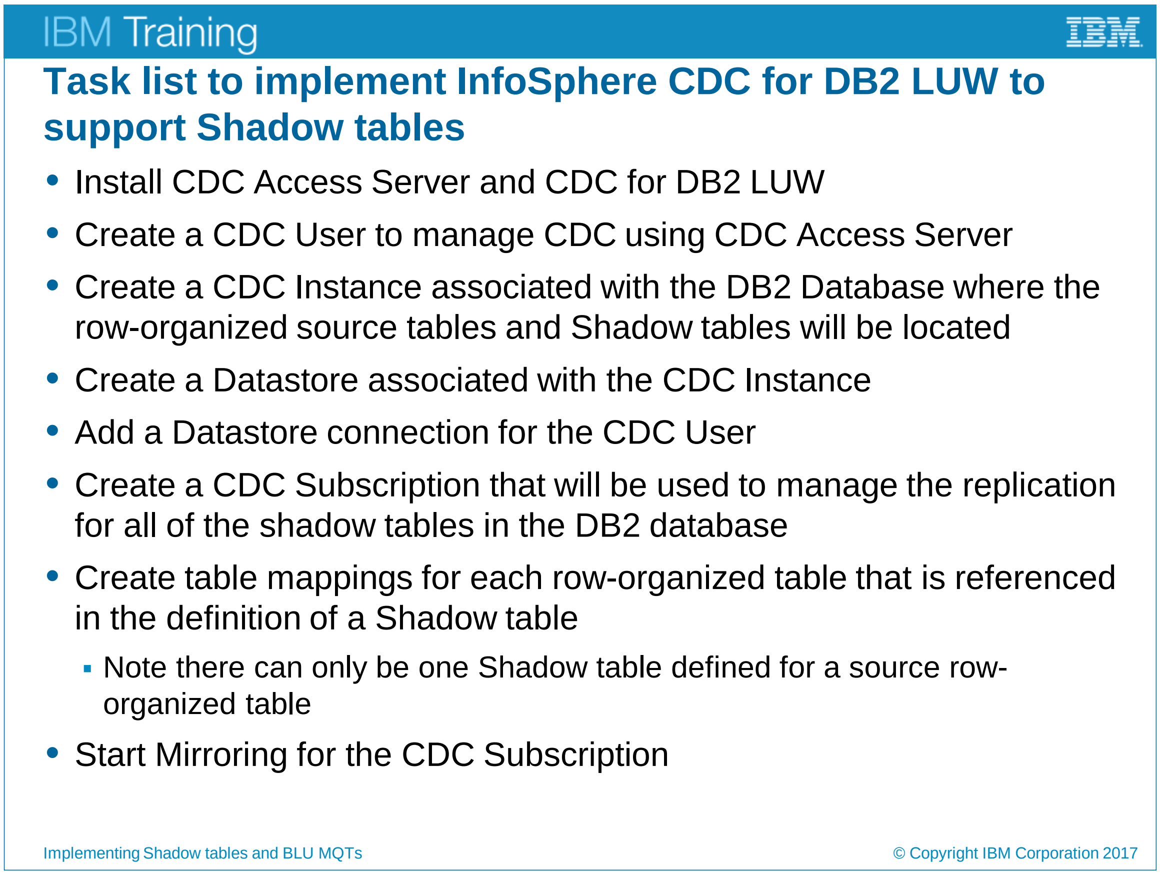
For shadow tables, you must create one single subscription that replicates all shadow tables in a database. Also, mark the subscription as persistent, which allows for better fault tolerance in situations where replication is disrupted.

Table mappings

Table mappings contain information on how individual tables (or columns of tables) are replicated from the source to the target datastores.

For shadow tables, choose standard replication with a one-to-one table mapping between a row-organized (source) table and the shadow (target) table. For the target table key, specify the unique index corresponding to the primary key of the shadow table to provide a one-to-one table mapping and performance improvements.

Before you add, modify, or delete table mappings that belong to a subscription, you must end replication.



IBM Training

Task list to implement InfoSphere CDC for DB2 LUW to support Shadow tables

- Install CDC Access Server and CDC for DB2 LUW
- Create a CDC User to manage CDC using CDC Access Server
- Create a CDC Instance associated with the DB2 Database where the row-organized source tables and Shadow tables will be located
- Create a Datastore associated with the CDC Instance
- Add a Datastore connection for the CDC User
- Create a CDC Subscription that will be used to manage the replication for all of the shadow tables in the DB2 database
- Create table mappings for each row-organized table that is referenced in the definition of a Shadow table
  - Note there can only be one Shadow table defined for a source row-organized table
- Start Mirroring for the CDC Subscription

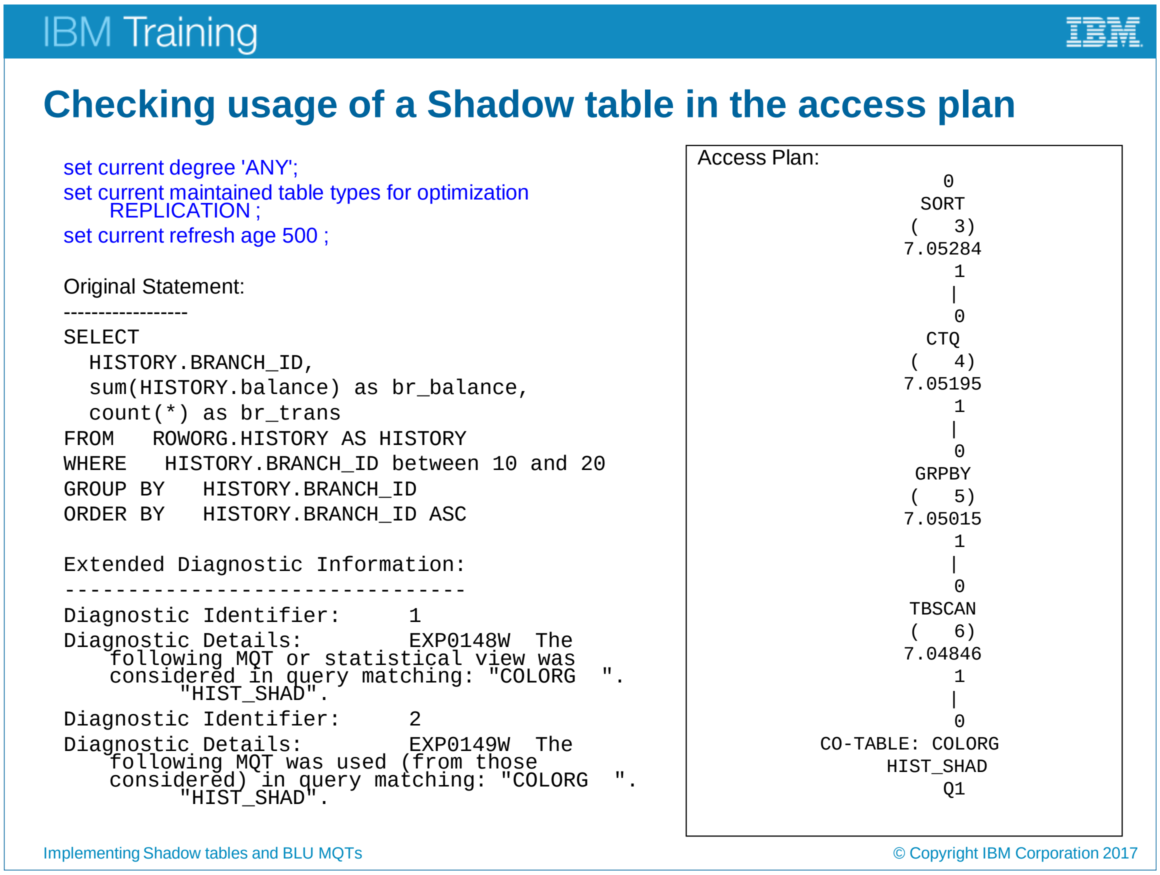
Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

Task list to implement InfoSphere CDC for DB2 LUW to support Shadow tables

The slide includes a list of tasks that you will perform to implement Infosphere CDC with a DB2 database to maintain a set of shadow tables.

The CDC software uses a user profile that contains a user name, the password and the role for the user, which sets limits on what the CDC user can perform.

When you create the CDC instance, you define the DB2 database user and password that DB2 will use to authorize the work performed by CDC in that database, including access to tables and running the LOAD utility.



IBM Training

Checking usage of a Shadow table in the access plan

```
set current degree 'ANY';
set current maintained table types for optimization
    REPLICATION ;
set current refresh age 500 ;

Original Statement:
------------------
SELECT
  HISTORY.BRANCH_ID,
  sum(HISTORY.balance) as br_balance,
  count(*) as br_trans
FROM   ROWORG.HISTORY AS HISTORY
WHERE   HISTORY.BRANCH_ID between 10 and 20
GROUP BY   HISTORY.BRANCH_ID
ORDER BY   HISTORY.BRANCH_ID ASC

Extended Diagnostic Information:
--------------------------------
Diagnostic Identifier:     1
Diagnostic Details:        EXP0148W  The
    following MQT or statistical view was
    considered in query matching: "COLORG  ".
          "HIST_SHAD".
Diagnostic Identifier:     2
Diagnostic Details:        EXP0149W  The
    following MQT was used (from those
    considered) in query matching: "COLORG  ".
          "HIST_SHAD".
```

```
Access Plan:
          0
         SORT
        (   3)
       7.05284
           1
           |
           0
          CTQ
        (   4)
       7.05195
           1
           |
           0
         GRPBY
        (   5)
       7.05015
           1
           |
           0
        TBSCAN
        (   6)
       7.04846
           1
           |
           0
  CO-TABLE: COLORG
      HIST_SHAD
          Q1
```

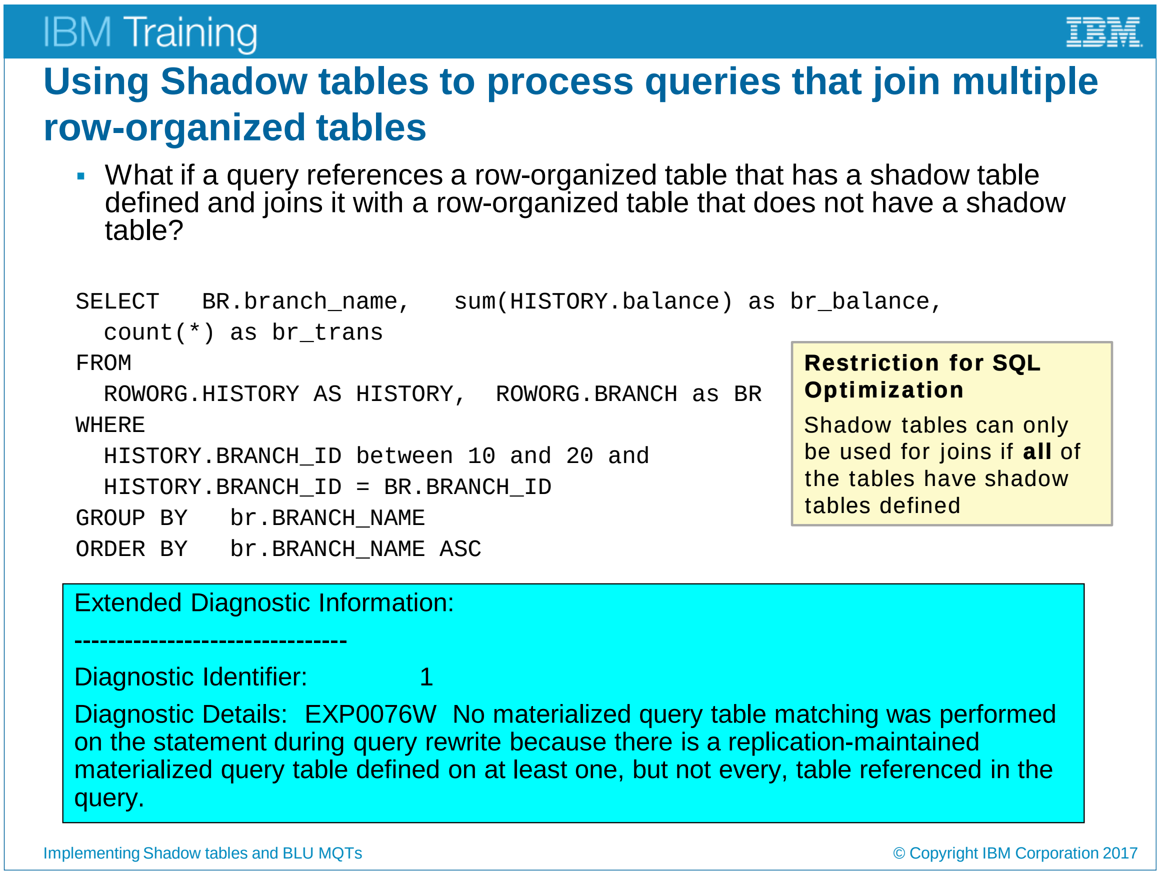
Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

Checking usage of a Shadow table in the access plan

The slide contains a series of SET CURRENT statements that an application could use to allow the DB2 optimizer to utilize a shadow table for an access plan.

The db2exfmt explain report shows the original SQL statement references the row-organized table ROWORG.HISTORY. The access plan graph shows that the shadow table, COLORG.HIST_SHAD.

The Diagnostic section of the explain report contains messages indicating that the shadow table was considered and selected for use in the access plan.



IBM Training

Using Shadow tables to process queries that join multiple row-organized tables

- What if a query references a row-organized table that has a shadow table defined and joins it with a row-organized table that does not have a shadow table?

```
SELECT   BR.branch_name,   sum(HISTORY.balance) as br_balance,
  count(*) as br_trans
FROM
  ROWORG.HISTORY AS HISTORY,  ROWORG.BRANCH as BR
WHERE
  HISTORY.BRANCH_ID between 10 and 20 and
  HISTORY.BRANCH_ID = BR.BRANCH_ID
GROUP BY   br.BRANCH_NAME
ORDER BY   br.BRANCH_NAME ASC
```

Restriction for SQL Optimization

Shadow tables can only be used for joins if **all** of the tables have shadow tables defined

```
Extended Diagnostic Information:
--------------------------------
Diagnostic Identifier:        1
Diagnostic Details:  EXP0076W  No materialized query table matching was performed
on the statement during query rewrite because there is a replication-maintained
materialized query table defined on at least one, but not every, table referenced in the
query.
```

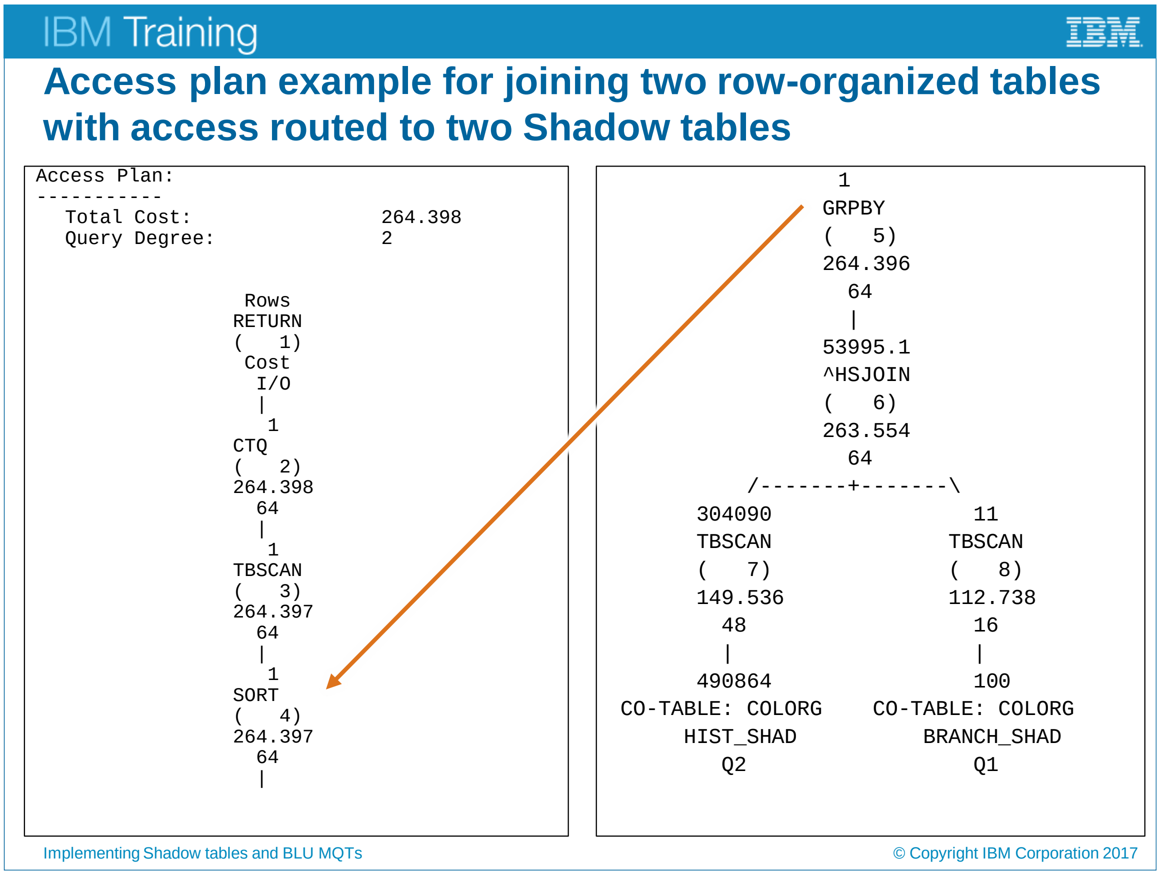
Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

Using Shadow tables to process queries that join multiple row-organized tables

With standard row-organized MQT tables, the MQT might be selected to perform a portion of the query processing and a table referenced in the SQL statement that is not included in the MQT definition could be joined with the MQT table in the access plan.

The sample SQL query shown references two tables, ROWORG.HISTORY that has an associated column-organized shadow table defined, and ROWORG.BRANCH a small row-organized table that does not have a matching shadow table.

The explain report for this SQL statement contains a warning message in the diagnostic section that describes the restriction that a replication-maintained MQT table (a shadow table) was detected during access plan generation, but since the SQL statement referenced a table that did not have a shadow table defined, the MQT matching was bypassed and the referenced row-organized tables were used.


IBM Training

Access plan example for joining two row-organized tables with access routed to two Shadow tables

```
Access Plan:
-----------
  Total Cost:                264.398
  Query Degree:              2


                Rows
               RETURN
               (   1)
                Cost
                 I/O
                 |
                  1
               CTQ
               (   2)
               264.398
                 64
                 |
                  1
               TBSCAN
               (   3)
               264.397
                 64
                 |
                  1
               SORT
               (   4)
               264.397
                 64
                 |
```

```
                      1
                    GRPBY
                    (   5)
                    264.396
                      64
                     |
                    53995.1
                    ^HSJOIN
                    (   6)
                    263.554
                      64
               /-------+-------\
         304090                  11
         TBSCAN                TBSCAN
         (   7)                (   8)
         149.536               112.738
           48                    16
           |                     |
         490864                  100
    CO-TABLE: COLORG      CO-TABLE: COLORG
         HIST_SHAD            BRANCH_SHAD
           Q2                    Q1
```

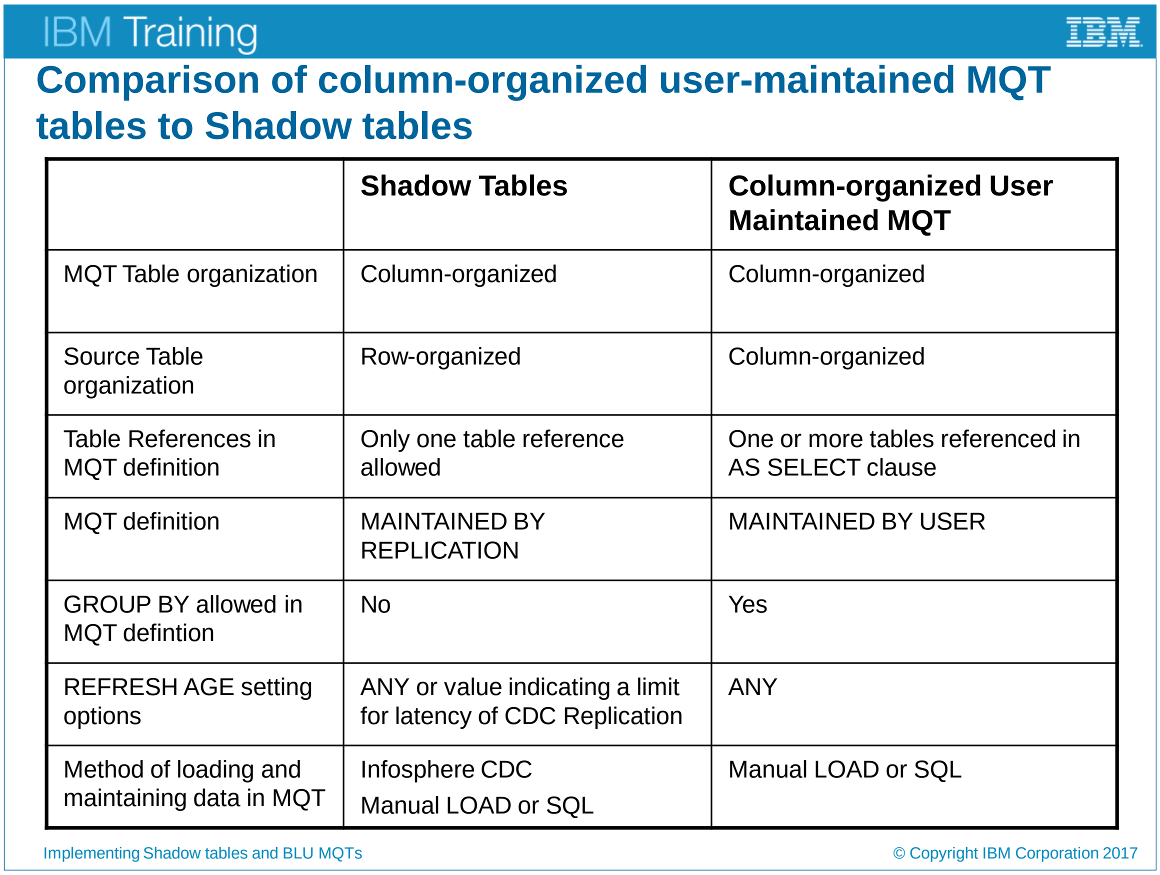
Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

Access plan example for joining two row-organized tables with access routed to two Shadow tables

The slide shows the access plan for the same SQL statement referenced in the previous slide can utilize two shadow tables if they are defined and meet the latency criteria set by the application.

The diagnostic section of the explain report would be similar to the following:

```
Extended Diagnostic Information:
--------------------------------
Diagnostic Identifier:  1
Diagnostic Details:    EXP0148W  The following MQT or statistical view
was considered in query matching: "COLORG  "."BRANCH_SHAD".
Diagnostic Identifier:  2
Diagnostic Details:    EXP0148W  The following MQT or statistical view
was considered in query matching: "COLORG  "."HIST_SHAD".
Diagnostic Identifier:  3
Diagnostic Details:    EXP0149W  The following MQT was used (from those
considered) in query matching: "COLORG  "."BRANCH_SHAD".
Diagnostic Identifier:  4
Diagnostic Details:    EXP0149W  The following MQT was used (from those
considered) in query matching: "COLORG  "."HIST_SHAD".
```



IBM Training

Comparison of column-organized user-maintained MQT tables to Shadow tables

	Shadow Tables	Column-organized User Maintained MQT
MQT Table organization	Column-organized	Column-organized
Source Table organization	Row-organized	Column-organized
Table References in MQT definition	Only one table reference allowed	One or more tables referenced in AS SELECT clause
MQT definition	MAINTAINED BY REPLICATION	MAINTAINED BY USER
GROUP BY allowed in MQT defintion	No	Yes
REFRESH AGE setting options	ANY or value indicating a limit for latency of CDC Replication	ANY
Method of loading and maintaining data in MQT	Infosphere CDC Manual LOAD or SQL	Manual LOAD or SQL

Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

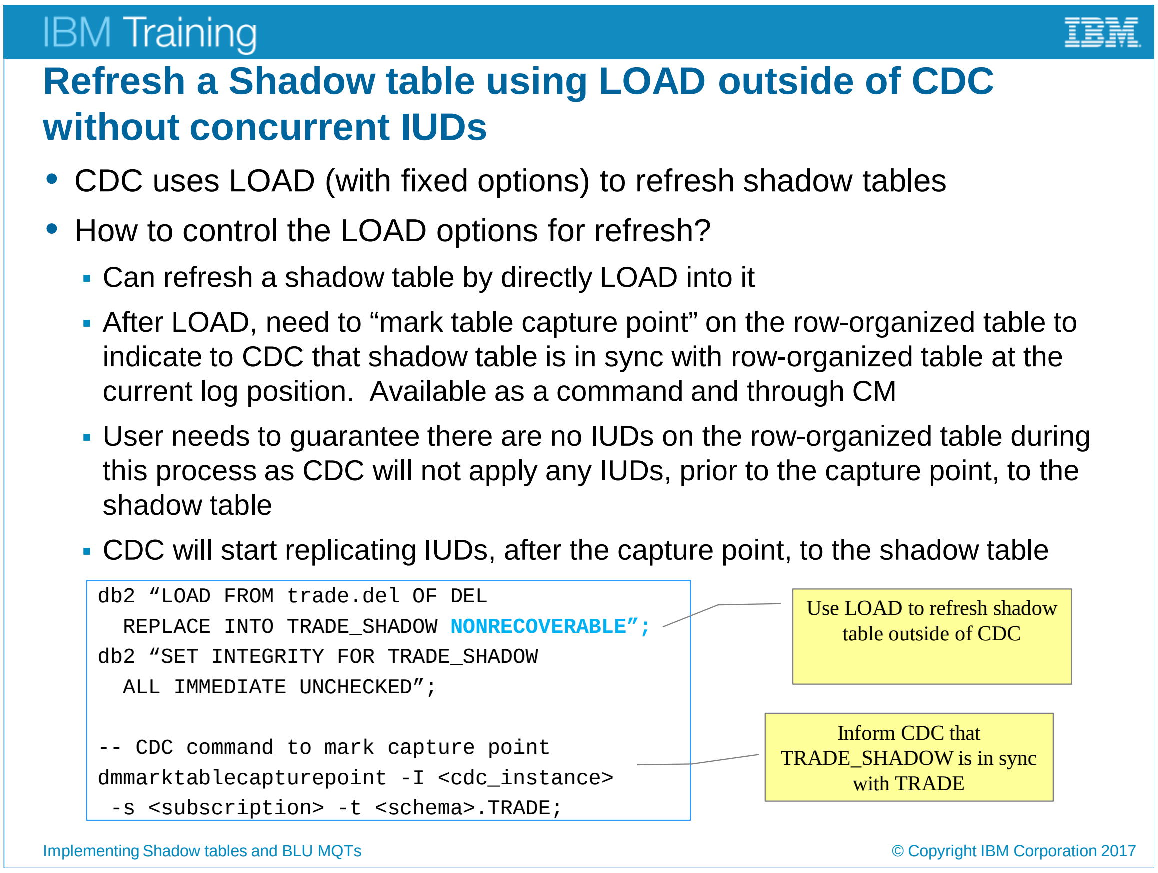
Comparison of column-organized user-maintained MQT tables to Shadow tables

The table compares some of the characteristics for the two types of column-organized MQT tables.

One key difference is that a shadow table is always based on a single row-organized table. A user maintained column-organized MQT table can reference one or more column-organized tables.

A shadow table MQT cannot contain a GROUP BY clause in the AS SELECT section of the MQT definition.

Latency-based routing of a query can only be performed using replication-maintained shadow tables. User maintained column-organized MQT tables must be accessed with a CURRENT REFRESH AGE set to ANY.



IBM Training

Refresh a Shadow table using LOAD outside of CDC without concurrent IUDs

- CDC uses LOAD (with fixed options) to refresh shadow tables
- How to control the LOAD options for refresh?
  - Can refresh a shadow table by directly LOAD into it
  - After LOAD, need to "mark table capture point" on the row-organized table to indicate to CDC that shadow table is in sync with row-organized table at the current log position. Available as a command and through CM
  - User needs to guarantee there are no IUDs on the row-organized table during this process as CDC will not apply any IUDs, prior to the capture point, to the shadow table
  - CDC will start replicating IUDs, after the capture point, to the shadow table

```
db2 "LOAD FROM trade.del OF DEL
  REPLACE INTO TRADE_SHADOW NONRECOVERABLE";
db2 "SET INTEGRITY FOR TRADE_SHADOW
  ALL IMMEDIATE UNCHECKED";

-- CDC command to mark capture point
dmmarktablecapturepoint -I <cdc_instance>
 -s <subscription> -t <schema>.TRADE;
```

Use LOAD to refresh shadow table outside of CDC

Inform CDC that TRADE_SHADOW is in sync with TRADE

Implementing Shadow tables and BLU MQTs

© Copyright IBM Corporation 2017

Refresh a Shadow table using LOAD outside of CDC without concurrent IUDs

When CDC refreshes a shadow table, it uses the DB2 LOAD command with a fixed set of options.

It is possible to directly LOAD into a shadow table outside of CDC, allowing the possibility to tailor the LOAD options. When directly running LOAD on the shadow table, it is important to use either NONRECOVERABLE or COPY YES to avoid putting the tablespace in backup pending.

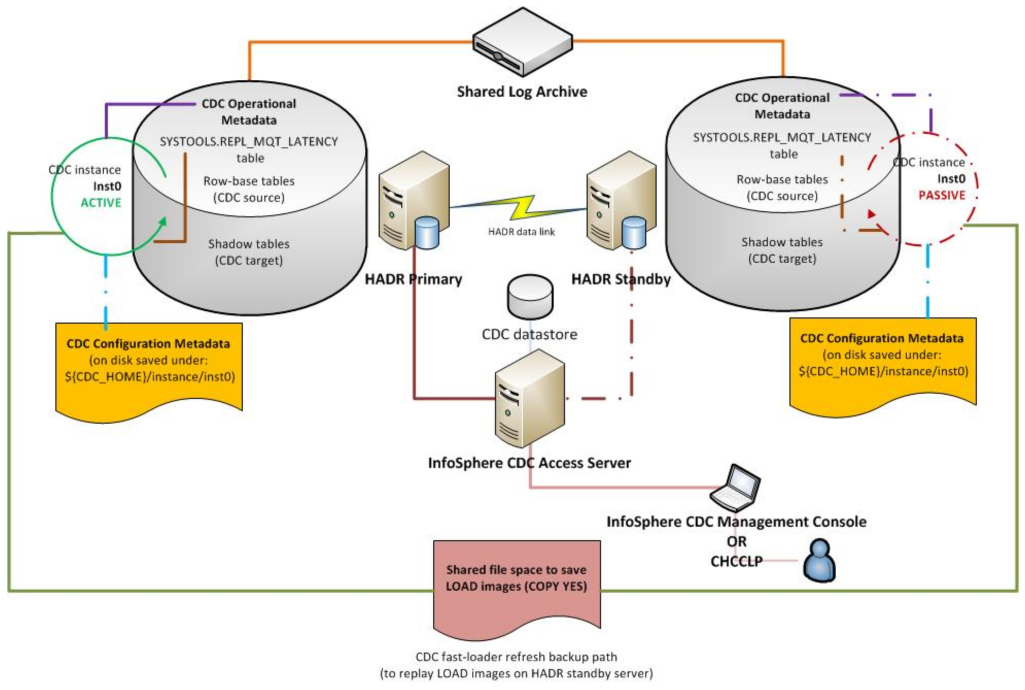
After the LOAD into the shadow table, you need to perform a "mark table capture point" on the row-organized table to inform CDC that the shadow table is sync with the row-organized table at the current log position.

Note CDC will not apply any Insert, Update or Deletes that occurs prior to this capture point to the shadow table. This implies that it is up to the user to guarantee that there are no IUDs on the row-organized table during this processing.

Once the mark capture point is performed, CDC will start to replicate any IUDs after the capture point to the shadow table when the subscription is started.

IBM Training

Using Shadow tables in a database with HADR Primary and Standby databases in use


Shared Log Archive
CDC Operational Metadata
SYSTOOLS.REPL_MQT_LATENCY table
Row-base tables (CDC source)
Shadow tables (CDC target)
CDC instance Inst0 ACTIVE
CDC instance Inst0 PASSIVE
HADR data link
HADR Primary
HADR Standby
CDC datastore
CDC Configuration Metadata (on disk saved under: ${CDC_HOME}/instance/inst0)
InfoSphere CDC Access Server
InfoSphere CDC Management Console OR CHCCLP
Shared file space to save LOAD images (COPY YES)
CDC fast-loader refresh backup path (to replay LOAD images on HADR standby server)

Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

Using Shadow tables in a database with HADR primary and standby databases in use

The use of shadow tables is supported for high availability disaster recovery (HADR) environments. However, there are a number of considerations for ensuring that latency-based routing is available in the event of a failover. For an HADR setup, you install and configure InfoSphere CDC on both a primary server and on the standby server. The InfoSphere CDC instance is active on the primary server and passive on the standby. HADR replicates data from the primary to the standby, including both source and shadow tables.

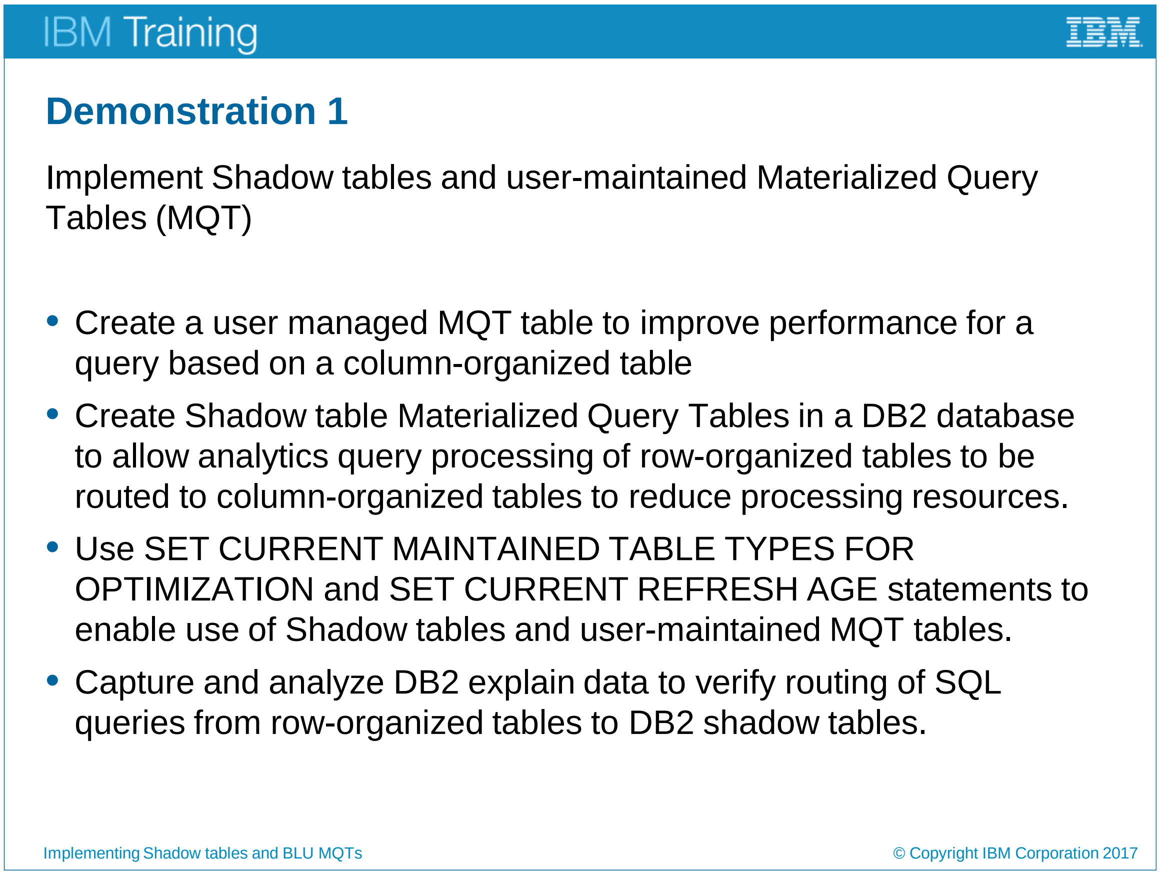
With InfoSphere CDC, there are two types of metadata that are used to synchronize the source and target tables. The operational metadata, which is information such as the instance signature and bookmarks, is stored in the database and is replicated by HADR to the standby servers.

The configuration metadata, which is information such as subscriptions and mapped tables, is stored in the CDC-installation-directory, so it is not replicated by HADR. Therefore, after you implement shadow tables in an HADR environment, any configuration changes that are made to the shadow tables on the primary are not reflected on the standby server.

You propagate configuration metadata changes by employing a scheduled InfoSphere CDC metadata backup-and-copy over the standby servers by using the **dmbackupmd** command.

You might also want to apply those changes to the standby servers immediately after they take place. For example, after any applying DDL statements that change the source row-organized table, the target shadow table, or both.

After a failover or role switch occurs and the HADR primary role has moved to a host where InfoSphere CDC was previously passive, you must manually start CDC on the new primary node. If the configuration metadata is not synchronized, you have to reapply any changes that occurred on the old primary server.



IBM Training

Demonstration 1

Implement Shadow tables and user-maintained Materialized Query Tables (MQT)

- Create a user managed MQT table to improve performance for a query based on a column-organized table
- Create Shadow table Materialized Query Tables in a DB2 database to allow analytics query processing of row-organized tables to be routed to column-organized tables to reduce processing resources.
- Use SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION and SET CURRENT REFRESH AGE statements to enable use of Shadow tables and user-maintained MQT tables.
- Capture and analyze DB2 explain data to verify routing of SQL queries from row-organized tables to DB2 shadow tables.

Implementing Shadow tables and BLU MQTs © Copyright IBM Corporation 2017

Demonstration 1: Implement Shadow tables and user-maintained Materialized Query Tables

Demonstration 1: Implement Shadow tables and user-maintained Materialized Query Tables

Purpose:
This demonstration will show you how to define and manage column-organized Shadow Tables for selected row-organized tables. We will also create a user maintained Materialized Query Table for a column-organized base table. You will not be configuring the Infosphere Change Data Capture based replication that would normally maintain the Shadow Table contents. The Shadow tables will be manually loaded. The DB2 explain tool will be used to verify routing the query processing to use the Shadow tables.

Task 1. Implement a user-maintained MQT for a column-organized table to improve query processing performance.

A User Maintained MQT will be created for a column-organized table. You will run a sample query that references the COLORG.HISTORY table

The demonstration is performed using command line Linux and DB2 commands.

1. From your Linux workstation, login with the userid of **inst450** and the password provided by the instructor (default ***ibm2blue***).

   Run the sample query using db2batch to report on the query performance.

   The sample query text is:

```
SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance,
count(*) as br_trans
   FROM COLORG.HISTORY AS HISTORY
   WHERE HISTORY.BRANCH_ID between 10 and 75
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;
```

   You will use the Linux Terminal session to enter DB2 commands.

2. Right-click the empty Linux desktop and select **Open in Terminal**.
3. Enter the following commands in the Linux terminal session:

   - **cd $HOME/db2blu**
   - **db2 connect to db2blu**
   - **db2batch -d db2blu -f histquery_mqt.sql -i complete -ISO CS | tee histmqt_bat1.txt**
   - **more histmqt_bat1.txt**

Review the db2batch report noting the following information:

Elapsed time for the query (SQL Statement Number 5):

For example:

```
SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance, count(*) as br_trans
   FROM COLORG.HISTORY AS HISTORY
   WHERE HISTORY.BRANCH_ID between 10 and 75
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;

BRANCH_ID BR_BALANCE                        BR_TRANS
--------- --------------------------------- -----------
       10                    2013883300.00       25939
       11                    2037791600.00       25746
       12                    2515666000.00       29556
       13                    3634933600.00       37235
       14                    2741741500.00       30871

* 66 row(s) fetched, 5 row(s) output.

* Prepare Time is:       0.189327 seconds
* Execute Time is:       0.226579 seconds
* Fetch Time is:         0.026361 seconds
* Elapsed Time is:       0.442267 seconds (complete)
```

The example elapsed time is 0.442267 seconds, with 0.189327 seconds of prepare time used to compile the access plan.

Look at the two queries that return performance metrics using the table function MON_GET_CONNECTION.

For example:

```
* SQL Statement Number 7:

SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS   TOTAL_L_READS
-------------------- -------------------- -------------------- --------------------
-
                 137                  325                  234
696

* 1 row(s) fetched, 1 row(s) output.
* Prepare Time is:       0.131173 seconds
* Execute Time is:       0.009095 seconds
* Fetch Time is:         0.000218 seconds
* Elapsed Time is:       0.140486 seconds (complete)

---------------------------------------------

* SQL Statement Number 8:

SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time,
total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

TOTAL_COL_TIME       TOTAL_COMPILE_TIME   POOL_READ_TIME       TOTAL_CPU_TIME
TOTAL_WAIT_TIME
-------------------- -------------------- -------------------- -------------------
- --------------------
                 704                  356                  336
243380                  882
```

Note the following statistics from your copy of the report:

TOTAL_L_READS:	(696 in the sample)
TOTAL_WAIT_TIME:	(882 in the sample)
POOL_READ_TIME:	(336 in the sample)

Now look at the access plan for the db2exfmt explain report.

4. In the terminal session, enter the following commands:
   - **db2exfmt -1 -d db2blu -o exumqt1.txt**
   - **more exumqt1.txt**

   The access plan in the report will be similar to the following:

```
Access Plan:
-----------
     Total Cost:        172.883
     Query Degree:            2


        Rows
       RETURN
       (   1)
        Cost
         I/O
         |
       87.7606
       LMTQ
       (   2)
       172.883
         53
         |
       87.7606
       CTQ
       (   3)
       172.862
         53
         |
       87.7606
       TBSCAN
       (   4)
       172.861
         53
         |
       87.7606
       SORT
       (   5)
       172.858
         53
         |
       87.7606
       GRPBY
       (   6)
       172.822
         53
         |
       430785
       TBSCAN
       (   7)
       166.105
         53
         |
       490864
 CO-TABLE: COLORG
       HISTORY
         Q1
```

Review the db2exfmt report noting the following information:

Total Estimated cost:	(172 in the sample)
Total I/O cost:	(53 in the sample)

The access plan uses a table scan to access the table COLORG.HISTORY.

Use the+ file *usermqt.ddl* to create a user maintained MQT, colorg.histumqt. The file contains the following DDL statements:

```
CREATE TABLE COLORG.HIST_UMQT
 ( BRANCH_ID, TELLER_ID, SBALANCE , SCOUNT )
 AS ( SELECT BRANCH_ID, TELLER_ID , SUM(BALANCE) AS SBALANCE,
 COUNT(*) AS SCOUNT FROM COLORG.HISTORY GROUP BY BRANCH_ID, TELLER_ID )
 DATA INITIALLY DEFERRED REFRESH DEFERRED MAINTAINED BY USER
 ORGANIZE BY COLUMN IN TSCOLD


SET INTEGRITY FOR COLORG.HIST_UMQT ALL IMMEDIATE UNCHECKED
```

5. In the terminal session, enter the following command:

   - **db2 -tvf usermqt.ddl**

   This MQT is defined to group and count the HISTORY data for each unique BRANCH_ID and TELLER_ID combination. It could be used for queries that summarize based on either of the two columns. We could create multiple MQT tables that summarize the data, for the specific use of certain queries.

   A user maintained MQT can be loaded using SQL statements or the LOAD utility. You cannot utilize a REFRESH TABLE statement like you would to populate a system maintained MQT. We will use a DB2 LOAD utility to load the user maintained MQT.

   The command file *loadumqt.sql* contains the necessary LOAD command to load the user maintained MQT using a declared cursor.

6. In the terminal session, enter the following command:

   - **db2 -tvf loadumqt.sql**

   The outywput should look similar to the following:

```
declare colhist1 cursor for SELECT BRANCH_ID, TELLER_ID , SUM(BALANCE) AS
SBALANCE, COUNT(*) AS SCOUNT FROM COLORG.HISTORY GROUP BY BRANCH_ID, TELLER_ID
order by BRANCH_ID, TELLER_ID
DB20000I  The SQL command completed successfully.

load from colhist1 of cursor  replace into COLORG.HIST_UMQT nonrecoverable
SQL3501W  The table space(s) in which the table resides will not be placed in
backup pending state since forward recovery is disabled for the database.

SQL1193I  The utility is beginning to load data from the SQL statement "
SELECT BRANCH_ID, TELLER_ID , SUM(BALANCE) AS SBALANCE, COUNT(*) ...".

SQL3500W  The utility is beginning the "ANALYZE" phase at time "05/17/2017
```

```
13:16:53.170654".

SQL3519W  Begin Load Consistency Point. Input record count = "0".

SQL3520W  Load Consistency Point was successful.

SQL3515W  The utility has finished the "ANALYZE" phase at time "05/17/2017
13:16:53.891892".

SQL3500W  The utility is beginning the "LOAD" phase at time "05/17/2017
13:16:53.892333".

SQL3110N  The utility has completed processing.  "34223" rows were read from
the input file.

SQL3519W  Begin Load Consistency Point. Input record count = "34223".

SQL3520W  Load Consistency Point was successful.

SQL3515W  The utility has finished the "LOAD" phase at time "05/17/2017
13:16:54.135136".

SQL3500W  The utility is beginning the "BUILD" phase at time "05/17/2017
13:16:54.137921".

SQL3213I  The indexing mode is "REBUILD".

SQL3515W  The utility has finished the "BUILD" phase at time "05/17/2017
13:16:54.220004".


Number of rows read         = 34223
Number of rows skipped      = 0
Number of rows loaded       = 34223
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 34223


runstats on table COLORG.HIST_UMQT and indexes all
DB20000I  The RUNSTATS command completed successfully.

SET INTEGRITY FOR COLORG.HIST_UMQT ALL IMMEDIATE UNCHECKED
DB20000I  The SQL command completed successfully.
```

You can run another db2batch execution to run the same SQL query, based on the table COLORG.HISTORY.

The file *histquery_mqt.sql* contains several statements necessary to set the conditions necessary for the DB2 compiler to allow routing the SQL processing to use the User Maintained MQT.

```
set current degree 'ANY';
set current maintained table types for optimization USER ;
set current refresh age ANY ;
```

7. Enter the following commands in the Linux terminal session:
   - **db2batch -d db2blu -f histquery_mqt.sql -i complete -ISO CS | tee histmqt_bat2.txt**
   - **more histmqt_bat2.txt**

   Review the db2batch report noting the following information:

   Elapsed time for the query (SQL Statement Number 5):

   For example:

```
* SQL Statement Number 5:

SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance, count(*) as br_trans
   FROM COLORG.HISTORY AS HISTORY
   WHERE HISTORY.BRANCH_ID between 10 and 75
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;

BRANCH_ID BR_BALANCE                        BR_TRANS
--------- --------------------------------- -----------
       10                     2013883300.00       25939
       11                     2037791600.00       25746
       12                     2515666000.00       29556
       13                     3634933600.00       37235
       14                     2741741500.00       30871

* 66 row(s) fetched, 5 row(s) output.

* Prepare Time is:       0.085893 seconds
* Execute Time is:       0.030773 seconds
* Fetch Time is:         0.004735 seconds
* Elapsed Time is:       0.121401 seconds (complete)
```

The example elapsed time is 0.121401 seconds.

Look at the two queries that return performance metrics using the table function MON_GET_CONNECTION.

For example:

```
* SQL Statement Number 7:

SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS   TOTAL_L_READS
-------------------- -------------------- -------------------- -------------------
-
                 122                  142                  140
404

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.000176 seconds
* Execute Time is:       0.008665 seconds
* Fetch Time is:         0.000225 seconds
* Elapsed Time is:       0.009066 seconds (complete)

---------------------------------------------

* SQL Statement Number 8:

SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time,
total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

TOTAL_COL_TIME       TOTAL_COMPILE_TIME   POOL_READ_TIME       TOTAL_CPU_TIME
TOTAL_WAIT_TIME
-------------------- -------------------- -------------------- -------------------
- --------------------
                 114                   89                   19
94193                  128
```

Note the following statistics from your copy of the report:

TOTAL_L_READS:	(404 in the sample)
TOTAL_WAIT_TIME:	(128 in the sample)
POOL_READ_TIME:	(19 in the sample)

Now look at the access plan for the db2exfmt explain report.

8. In the terminal session, enter the following commands:
   - **db2exfmt -1 -d db2blu -o exumqt2.txt**
   - **more exumqt2.txt**

   The access plan in the report will be similar to the following:

```
Access Plan:
-----------
     Total Cost:         166.224
     Query Degree:             2


        Rows
       RETURN
       (   1)
        Cost
         I/O
         |
         66
       LMTQ
       (   2)
       166.224
         23
         |
         66
       CTQ
       (   3)
       166.206
         23
         |
         66
       TBSCAN
       (   4)
       166.205
         23
         |
         66
       SORT
       (   5)
       166.203
```

```
          23
          |
          66
        GRPBY
        (   6)
        166.177
          23
          |
        22587.2
        TBSCAN
        (   7)
        165.823
          23
          |
         34223
  CO-TABLE: COLORG
       HIST_UMQT
          Q1
```

Review the db2exfmt report noting the following information:

Total Estimated cost:	(166 in the sample)
Total I/O cost:	(23 in the sample)

The access plan uses a table scan to access the user maintained MQT table COLORG.HIST_UMQT.

Task 2. Create Shadow tables for several row-organized tables to improve analytics query processing.

There is an existing DB2 database named **BLUTRAN** that has a set of Row-organized tables that are being used to support some OLTP applications. You will use some sample analytics queries in a file named *testq1.sql* that produces summarized reports using the row-organized tables. You will use **db2batch** to execute these queries as a group.

The file *testq1.sql* contains the following SQL statements:

```
SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance,
count(*) as br_trans
   FROM ROWORG.HISTORY AS HISTORY
   WHERE HISTORY.BRANCH_ID between 10 and 20
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;

SELECT acct.acct_grp, sum(acct.balance) as grp_balance, count(*) as
grp_trans
   FROM ROWORG.acct as acct
   WHERE acct.acct_grp > 50
   GROUP BY acct.acct_grp
   ORDER BY acct.acct_grp ASC ;

SELECT BR.branch_name , sum(HISTORY.balance) as br_balance, count(*)
as br_trans
   FROM ROWORG.HISTORY AS HISTORY, roworg.branch as BR
   WHERE HISTORY.BRANCH_ID between 10 and 20
   and HISTORY.BRANCH_ID = BR.BRANCH_ID
   GROUP BY br.BRANCH_NAME
   ORDER BY br.BRANCH_NAME ASC ;

select
varchar(tabname,20) as table,
varchar(tabschema,12) as schema,
rows_read,
table_scans, num_columns_referenced,
section_exec_with_col_references,
( num_columns_referenced / section_exec_with_col_references ) as
avg_columns_persql
from table(mon_get_table('ROWORG',NULL,-2)) as t1
order by tabname ;
```

You will use the Linux Terminal session to enter DB2 commands.

1. Right-click the empty Linux desktop and select **Open in Terminal**.
2. Enter the following commands in the Linux terminal session:
   - **cd $HOME/blutran**
   - **db2 terminate**
   - **db2 connect to blutran**
   - **db2batch -d blutran -f testq1.sql -i complete -iso cs | tee shad_bat1.txt**
   - **more shad_bat1.txt**

   The db2batch run also included a SQL query that shows how many columns are being accessed per query for each table in the schema ROWORG.

   Review the result from the SQL query that uses MON_GET_TABLE to report on table accesses.

   For example:

```
TABLE                SCHEMA       ROWS_READ            TABLE_SCANS
NUM_COLUMNS_REFERENCED SECTION_EXEC_WITH_COL_REFERENCES AVG_COLUMNS_PERSQL
-------------------- ------------ -------------------- -------------------- ------
---------------- ------------------------------ -------------------
ACCT                 ROWORG                    1000000                    2
2                               1                    2
BRANCH               ROWORG                         11                    0
2                               1                    2
HISTORY              ROWORG                     627152                    0
4                               2                    2


* 3 row(s) fetched, 3 row(s) output.
```

   This query result lists the three row-organized tables with the number of rows read and the average number of column references per SQL execution. The small number of column references per table suggests that using a column-organized table could efficiently handle these SQL statements.

   We want to be able to continue to run the same SQL text, so we can implement shadow table MTQ tables to transparently route the processing to the column-organized shadow tables when the DB2 optimizer considers the shadow table a more efficient method to produce the result.

   You will not fully implement the Shadow tables for this exercise. The supporting objects in Infosphere Change Data Capture will not be created. We can simply create the Shadow table Materialized Query tables and manually load the necessary data.

   You will start by defining two Shadow Tables to provide column-organized copies of the two larger tables, ACCT and HISTORY.

DB2 uses a special system table to communicate latency information for shadow tables. You will create the table SYSTOOLS.REPL_MQT_LATENCY. The procedure SYSINSTALLOBJECTS can be used to create this table for the database BLUTRAN. The file create_replmqt.ddl contains the procedure call.

3. Enter the following command in the Linux terminal session:

    - **db2 -tvf create_replmqt.ddl**

    The output will be similar to the following:

```
call sysinstallobjects('REPL_MQT','C','TSWORK',NULL)

  Return Status = 0
```

You will create the Shadow Table, COLORG.HIST_SHAD for the row-organized table ROWORG.HISTORY. The clause MAINTAINED BY REPLICATION designates this MQT to be a shadow table. Other clauses REFRESH DEFERRED and ORGANIZE BY COLUMN are required. Use the file *create_hist_shad.ddl* to create the Shadow table. The file also contains the SET INTEGRITY with the IMMEDIATE UNCHECKED clause and the ALTER TABLE statement to define the required primary key for the shadow table.

4. Enter the following commands in the Linux terminal session:

    - **db2 connect to blutran**
    - **db2 -tvf create_hist_shadow.ddl**

    The output will be similar to the following:

```
create table colorg.hist_shad as (select * from roworg.history ) data initially
deferred refresh deferred enable query optimization maintained by replication
organize by column in tsshadd index in tsshadi
DB20000I  The SQL command completed successfully.

set integrity for colorg.hist_shad all immediate unchecked
DB20000I  The SQL command completed successfully.

alter table colorg.hist_shad add constraint hist_shad_pk primary key (acct_id,
tstmp)
DB20000I  The SQL command completed successfully.
```

Note that this shadow table is based on a 'SELECT *' SQL statement so all of the columns from the source table will be replicated.

You will create the Shadow Table, COLORG.ACCT_SHAD for the row-organized table ROWORG.ACCT. The clauses MAINTAINED BY REPLICATION, REFRESH DEFERRED and ORGANIZE BY COLUMN are included. Use the file *create_acct_shadow*.ddl to create the Shadow table. The file also contains the SET INTEGRITY with the IMMEDIATE UNCHECKED clause and the ALTER TABLE statement to define the required primary key for the shadow table.

5. Enter the following commands in the Linux terminal session:
   - **db2 connect to blutran**
   - **db2 -tvf create_acct_shad.ddl**

   The output will be similar to the following:

```
CREATE TABLE COLORG.ACCT_SHAD ( ACCT_ID, ACCT_GRP, BALANCE ) AS ( SELECT ACCT_ID,
ACCT_GRP, BALANCE FROM ROWORG.ACCT ) DATA INITIALLY DEFERRED REFRESH DEFERRED
MAINTAINED BY REPLICATION ORGANIZE BY COLUMN IN TSSHADD INDEX IN TSSHADI
DB20000I  The SQL command completed successfully.

SET INTEGRITY FOR COLORG.ACCT_SHAD ALL IMMEDIATE UNCHECKED
DB20000I  The SQL command completed successfully.

ALTER TABLE COLORG.ACCT_SHAD ADD CONSTRAINT ACCT_SHAD_PK PRIMARY KEY ( ACCT_ID )
DB20000I  The SQL command completed successfully.
```

   This shadow table is based on a SELECT SQL statement that only includes the columns that we expect to use in analytics query processing.

   For the purpose of this test you will manually load data into the shadow tables from the row-organized tables.

   Use the file *loadacct_shad.sql* to load the shadow table for the ACCT tables.

6. Enter the following command in the Linux terminal session:
   - **db2 -tvf loadacct_shad.sql**

   The output will be similar to the following:

```
declare rowacct cursor for SELECT ACCT_ID, ACCT_GRP, BALANCE FROM ROWORG.ACCT
DB20000I  The SQL command completed successfully.

load from rowacct of cursor  replace into COLORG.ACCT_SHAD nonrecoverable
SQL1193I  The utility is beginning to load data from the SQL statement "
SELECT ACCT_ID, ACCT_GRP, BALANCE FROM ROWORG.ACCT   ".
```

```
SQL3500W  The utility is beginning the "ANALYZE" phase at time "05/17/2017
15:25:43.643132".

SQL3519W  Begin Load Consistency Point. Input record count = "0".

SQL3520W  Load Consistency Point was successful.

SQL3515W  The utility has finished the "ANALYZE" phase at time "05/17/2017
15:25:47.710985".

SQL3500W  The utility is beginning the "LOAD" phase at time "05/17/2017
15:25:47.713093".

SQL3110N  The utility has completed processing.  "1000000" rows were read from
the input file.

SQL3519W  Begin Load Consistency Point. Input record count = "1000000".

SQL3520W  Load Consistency Point was successful.

SQL3515W  The utility has finished the "LOAD" phase at time "05/17/2017
15:25:50.961321".

SQL3500W  The utility is beginning the "BUILD" phase at time "05/17/2017
15:25:50.965392".

SQL3213I  The indexing mode is "REBUILD".

SQL3515W  The utility has finished the "BUILD" phase at time "05/17/2017
15:25:52.229544".


Number of rows read         = 1000000
Number of rows skipped      = 0
Number of rows loaded       = 1000000
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 1000000


runstats on table COLORG.ACCT_SHAD and indexes all
DB20000I  The RUNSTATS command completed successfully.

SET INTEGRITY FOR COLORG.ACCT_SHAD ALL IMMEDIATE UNCHECKED
DB20000I  The SQL command completed successfully.
```

Use the file *loadhist_shad.sql* to load the shadow table for the HISTORY tables.

7. Enter the following command in the Linux terminal session:
   - **db2 -tvf loadhist_shad.sql**

   The output will be similar to the following:

```
declare rowhist cursor for SELECT * from roworg.history
DB20000I  The SQL command completed successfully.

load from rowhist of cursor  replace into colorg.hist_shad nonrecoverable
SQL1193I  The utility is beginning to load data from the SQL statement "
SELECT * from roworg.history   ".

SQL3500W  The utility is beginning the "ANALYZE" phase at time "05/17/2017
15:26:43.503008".

SQL3519W  Begin Load Consistency Point. Input record count = "0".

SQL3520W  Load Consistency Point was successful.

SQL3515W  The utility has finished the "ANALYZE" phase at time "05/17/2017
15:26:47.599957".

SQL3500W  The utility is beginning the "LOAD" phase at time "05/17/2017
15:26:47.601905".

SQL3110N  The utility has completed processing.  "490864" rows were read from
the input file.

SQL3519W  Begin Load Consistency Point. Input record count = "490864".

SQL3520W  Load Consistency Point was successful.

SQL3515W  The utility has finished the "LOAD" phase at time "05/17/2017
15:26:51.127138".

SQL3500W  The utility is beginning the "BUILD" phase at time "05/17/2017
15:26:51.131763".

SQL3213I  The indexing mode is "REBUILD".

SQL3515W  The utility has finished the "BUILD" phase at time "05/17/2017
15:26:52.842493".


Number of rows read         = 490864
Number of rows skipped      = 0
Number of rows loaded       = 490864
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 490864


runstats on table colorg.hist_shad and indexes all
DB20000I  The RUNSTATS command completed successfully.

SET INTEGRITY FOR colorg.hist_shad ALL IMMEDIATE UNCHECKED
DB20000I  The SQL command completed successfully.
```

You now can run SQL queries with references to the column-organized tables and generate the explain data to see if the associated column-organized shadow tables are referenced in the access plans.

The first SQL query produces a summary report based on the table ROWORG.HISTORY. The SQL statements are in a file named *exp_shadq1.sql*, which contains the following SQL statements:

```
set current degree 'ANY';

set current explain mode explain;
set current maintained table types for optimization replication ;
set current refresh age any ;



SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance, count(*) as br_trans
   FROM ROWORG.HISTORY AS HISTORY
   WHERE HISTORY.BRANCH_ID between 10 and 20
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;

set current explain mode no;
```

In order for the DB2 optimizer to consider using the shadow tables to replace the row-organized tables in access plans, a number of conditions must be met. The SET CURRENT statements address these conditions.

- SET CURRENT DEGREE ‘ANY’ is necessary for the parallel processing used to access column-organized tables.
- SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION REPLICATION specifies that only MAINTAINED BY REPLICATION types of MQT tables can be considered during SQL compilation. This setting is required when the Latency-based routing for shadow tables is desired.
- SET CURRENT REFRESH AGE ANY allows the shadow table to be referenced without checking for a specific latency limit.

8. Enter the following commands in the Linux terminal session:
   - **db2 terminate**
   - **db2 connect to blutran**
   - **db2 -tvf exp_shadq1.sql**
   - **db2exfmt -1 -d blutran -o ex_hist_shadow.txt**
   - **more ex_hist_shadow.txt**

   Review the access plan and extended diagnostic sections of the explain report.

```
Access Plan:
-----------
     Total Cost:        167.095
     Query Degree:            2


         Rows
        RETURN
        (   1)
         Cost
          I/O
          |
          11
        LMTQ
        (   2)
        167.095
          53
          |
          11
        CTQ
        (   3)
        167.083
          53
          |
          11
        TBSCAN
        (   4)
        167.082
          53
          |
          11
        SORT
        (   5)
        167.082
          53
          |
          11
        GRPBY
        (   6)
        167.078
          53
          |
        53995.1
        TBSCAN
        (   7)
        166.235
          53
          |
        490864
  CO-TABLE: COLORG
       HIST_SHAD
          Q1


Operator Symbols :
------------------

   Symbol      Description
   ---------   ------------------------------------------
    ATQ      : Asynchrony
    BTQ      : Broadcast
    CTQ      : Column-organized data
    DTQ      : Directed
    LTQ      : Intra-partition parallelism
    MTQ      : Merging (sorted)
    STQ      : Scatter
   RCTQ      : Column-organized data with row as the source
    XTQ      : XML aggregation
     TQ*     : Listener


Extended Diagnostic Information:
--------------------------------

Diagnostic Identifier:  1
Diagnostic Details:     EXP0148W  The following MQT or statistical view was
                 considered in query matching: "COLORG  ".
                 "HIST_SHAD".
Diagnostic Identifier:  2
Diagnostic Details:     EXP0149W  The following MQT was used (from those
                 considered) in query matching: "COLORG  ".
                 "HIST_SHAD".
```

The access plan includes the shadow table, as noted in the diagnostic information.

Next you will explain a SQL query that references the row-organized table ROWORG.ACCT. The SQL statements are in a file named *exp_shad_acct.sql*, which contains the following SQL statements:

```
set current degree 'ANY';

set current explain mode explain;
set current maintained table types for optimization = REPLICATION,USER   ;
set current refresh age any ;



SELECT acct.acct_grp, sum(acct.balance) as grp_balance, count(*) as grp_trans
   FROM ROWORG.acct as acct
   WHERE acct.acct_grp > 50
   GROUP BY acct.acct_grp
   ORDER BY acct.acct_grp ASC ;

set current explain mode no;
```

The SET CURRENT statements address these conditions.

- SET CURRENT DEGREE 'ANY' is necessary for the parallel processing used to access column-organized tables.
- SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION REPLICATION,USER specifies that either MAINTAINED BY REPLICATION or MAINTAINED BY USER types of MQT tables can be considered during SQL compilation. This setting is allowed when the Latency-based routing for shadow tables is NOT requested.
- SET CURRENT REFRESH AGE ANY is used when Latency-based routing is not required to produce the application query results.

9. Enter the following commands in the Linux terminal session:
   - **db2 -tvf exp_shad_acct.sql**
   - **db2exfmt -1 -d blutran -o ex_acct_shadow.txt**
   - **more ex_acct_shadow.txt**

   Review the access plan and extended diagnostic sections of the explain report.

```
Access Plan:
-----------
     Total Cost:        335.304
     Query Degree:            2
```

```
         Rows
        RETURN
        (   1)
         Cost
          I/O
          |
        949.049
        LMTQ
        (   2)
        335.304
          130
          |
        949.049
        CTQ
        (   3)
        335.188
          130
          |
        949.049
        TBSCAN
        (   4)
        335.183
          130
          |
        949.049
        SORT
        (   5)
        335.152
          130
          |
        949.049
        GRPBY
        (   6)
        334.653
          130
          |
        949049
        TBSCAN
        (   7)
        319.847
          130
          |
         1e+06
  CO-TABLE: COLORG
       ACCT_SHAD
          Q1
```

```
Operator Symbols :
------------------

   Symbol      Description
   ---------   ------------------------------------------
    ATQ      : Asynchrony
    BTQ      : Broadcast
    CTQ      : Column-organized data
    DTQ      : Directed
    LTQ      : Intra-partition parallelism
    MTQ      : Merging (sorted)
    STQ      : Scatter
   RCTQ      : Column-organized data with row as the source
    XTQ      : XML aggregation
     TQ*     : Listener


Extended Diagnostic Information:
--------------------------------

Diagnostic Identifier:  1
Diagnostic Details:     EXP0148W  The following MQT or statistical view was
                 considered in query matching: "COLORG  ".
                 "ACCT_SHAD".
Diagnostic Identifier:  2
Diagnostic Details:     EXP0149W  The following MQT was used (from those
                 considered) in query matching: "COLORG  ".
                 "ACCT_SHAD".
```

The access plan includes the shadow table, as noted in the diagnostic information.

Next you will produce the explain report for a SQL statement that joins two row-organized tables ROWORG.HISTORY and ROWORG.BRANCH. The SQL statements are in a file named *exp_shadjoin.sql*, which contains the following SQL statements:

```
set current degree 'ANY';

set current explain mode explain;
set current maintained table types for optimization replication ;
set current refresh age any ;


SELECT BR.branch_name , sum(HISTORY.balance) as br_balance, count(*) as br_trans
   FROM ROWORG.HISTORY AS HISTORY, roworg.branch as BR
   WHERE HISTORY.BRANCH_ID between 10 and 20
   and HISTORY.BRANCH_ID = BR.BRANCH_ID
   GROUP BY br.BRANCH_NAME
   ORDER BY br.BRANCH_NAME ASC ;

set current explain mode no;
```

10. Enter the following commands in the Linux terminal session:
    - **db2 -tvf exp_shadjoin.sql**
    - **db2exfmt -1 -d blutran -o ex_join_shadow1.txt**
    - **more ex_join_shadow1.txt**

    Review the access plan and extended diagnostic sections of the explain report. Your access plan may be different. For example, it might show a hash join (HSJOIN) instead of a nested loop join (NLJOIN). However, your Diagnostic Details should be the same as shown below.

```
Access Plan:
-----------
     Total Cost:         1754.01
     Query Degree:             2


                           Rows
                          RETURN
                          (   1)
                           Cost
                            I/O
                            |
                             1
                          GRPBY
                          (   2)
                          1754.01
                          841.734
                            |
                             1
                          LMTQ
                          (   3)
                          1754.01
                          841.734
                            |
                             1
                          GRPBY
                          (   4)
                           1754
                          841.734
                            |
                          53995.1
                          NLJOIN
                          (   5)
                          1750.96
                          841.734
                 /----------+-----------\
               11                      4908.64
             TBSCAN                    FETCH
             (   6)                    (  10)
             7.05915                   1607.57
                1                      794.554
               |                     /---+----\
               11                4908.64      490864
             SORT                RIDSCN   TABLE: ROWORG
             (   7)              (  11)       HISTORY
             7.05898             12.8707        Q2
                1                 1.43
               |                   |
               11                4908.64
             FETCH               SORT
             (   8)              (  12)
             7.05829             12.8705
                1                 1.43
          /----+----\              |
        11           100        4908.64
      IXSCAN    TABLE: ROWORG    IXSCAN
      (   9)        BRANCH       (  13)
     0.0101862        Q1         11.974
         0                        1.43
        |                          |
        100                      490864
  INDEX: SYSIBM              INDEX: ROWORG
SQL170327113013710               HISTIX2
        Q1                         Q2

Diagnostic Identifier:  1
Diagnostic Details:     EXP0076W  No materialized query table matching was
                performed on the statement during query rewrite
                because there is a shadow table defined on at least
                one, but not every, row-organized table referenced
                in the query, or a row-organized table has a
                REFRESH IMMEDIATE materialized query defined on it.
                The following row-organized table that is used in
                the query does not have a shadow table defined on
                it, or the following REFRESH IMMEDIATE materialized
                query table is defined on a row-organized table
                that is used in the query: "ROWORG.BRANCH".
```

```
Diagnostic Identifier:  2
Diagnostic Details:     EXP0080W  The current usage of the statement or the
                 statement containing update, delete or insert or
                 constructs like sampling limits MQT matching.
```

The access plan shows that the base row-organized tables will be joined and the shadow table COLORG.HIST_SHAD was not used. The estimated cost for this query is about 1800, using the row-organized tables.

The diagnostic information in the explain report shows that the DB2 optimizer did not perform MQT matching because the table ROWORG.BRANCH did not have a shadow table defined.

You will create another Shadow Table, COLORG.BRANCH_SHAD for the row-organized table ROWORG.BRANCH, to allow the shadow table COLORG.HIST_SHAD to be utilized when a SQL query joins the BRANCH and HISTORY tables. Use the file *create_branch_shad.ddl* to create the Shadow table.

The file also contains the SET INTEGRITY with the IMMEDIATE UNCHECKED clause and the ALTER TABLE statement to define the required primary key for the shadow table.

11. Enter the following commands in the Linux terminal session:
    - **db2 connect to blutran**
    - **db2 -tvf create_branch_shadow.ddl**

    The output should be similar to the following:

```
CREATE TABLE COLORG.BRANCH_SHAD ( BRANCH_ID, BRANCH_NAME, BALANCE )
AS ( SELECT BRANCH_ID, BRANCH_NAME, BALANCE FROM ROWORG.BRANCH  )
DATA INITIALLY DEFERRED REFRESH DEFERRED MAINTAINED BY REPLICATION
ORGANIZE BY COLUMN IN TSSHADD INDEX IN TSSHADI
DB20000I  The SQL command completed successfully.

SET INTEGRITY FOR COLORG.BRANCH_SHAD ALL IMMEDIATE UNCHECKED
DB20000I  The SQL command completed successfully.

ALTER TABLE COLORG.BRANCH_SHAD ADD CONSTRAINT BRANCH_SHAD_PK PRIMARY
KEY ( BRANCH_ID )
DB20000I  The SQL command completed successfully.
```

    Note that this shadow table is based on a SELECT SQL statement that lists a few columns that we think will be needed for analytics queries.

    You will manually load the data from the base row-organized table into the new shadow table. In a production system, you would use INFOSPHERE CDC software to automatically replicate the data.

Use the file *loadbranch_shad.sql* to load the shadow table for the BRANCH table.

12. Enter the following command in the Linux terminal session:

    - **db2 -tvf loadbranch_shad.sql**

    The output will be similar to the following:

```
declare rowbranch cursor for SELECT BRANCH_ID, BRANCH_NAME, BALANCE FROM
ROWORG.BRANCH
DB20000I  The SQL command completed successfully.

load from rowbranch of cursor  replace into COLORG.BRANCH_SHAD nonrecoverable
SQL1193I  The utility is beginning to load data from the SQL statement "
SELECT BRANCH_ID, BRANCH_NAME, BALANCE FROM ROWORG.BRANCH   ".

SQL3500W  The utility is beginning the "ANALYZE" phase at time "05/17/2017
15:44:59.430483".

SQL3519W  Begin Load Consistency Point. Input record count = "0".

SQL3520W  Load Consistency Point was successful.

SQL3515W  The utility has finished the "ANALYZE" phase at time "05/17/2017
15:44:59.573583".

SQL3500W  The utility is beginning the "LOAD" phase at time "05/17/2017
15:44:59.574048".

SQL3110N  The utility has completed processing.  "100" rows were read from the
input file.

SQL3519W  Begin Load Consistency Point. Input record count = "100".

SQL3520W  Load Consistency Point was successful.

SQL3515W  The utility has finished the "LOAD" phase at time "05/17/2017
15:44:59.678263".

SQL3500W  The utility is beginning the "BUILD" phase at time "05/17/2017
15:44:59.681711".

SQL3213I  The indexing mode is "REBUILD".

SQL3515W  The utility has finished the "BUILD" phase at time "05/17/2017
15:44:59.756453".
```

```
Number of rows read         = 100
Number of rows skipped      = 0
Number of rows loaded       = 100
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 100


runstats on table COLORG.BRANCH_SHAD and indexes all
DB20000I  The RUNSTATS command completed successfully.

SET INTEGRITY FOR COLORG.BRANCH_SHAD ALL IMMEDIATE UNCHECKED
DB20000I  The SQL command completed successfully.
```

Now that you have a shadow table created for ROWORG.BRANCH you will generate the explain report for the SQL statement again that joins two row-organized tables ROWORG.HISTORY and ROWORG.BRANCH. You will collect statistics on the new shadow table to improve DB2 compiler information for estimating access to the shadow table.

13. Enter the following commands in the Linux terminal session:
    - **db2 connect to blutran**
    - **db2 runstats on table colorg.branch_shad**
    - **db2 -tvf exp_shadjoin.sql**
    - **db2exfmt -1 -d blutran -o ex_join_shadow2.txt**
    - **more ex_join_shadow2.txt**

    Review the access plan and extended diagnostic sections of the explain report.

```
Access Plan:
-----------
     Total Cost:        276.053
     Query Degree:            2


                 Rows
                RETURN
                (   1)
                 Cost
                  I/O
                  |
                   1
                CTQ
                (   2)
                276.053
                  69
                  |
                   1
                TBSCAN
                (   3)
                276.053
                  69
                  |
                   1
                SORT
                (   4)
                276.053
                  69
                  |
                   1
                GRPBY
                (   5)
                276.052
                  69
                  |
                53995.1
                ^HSJOIN
                (   6)
                275.209
                  69
          /-------+-------\
      53995.1               11
      TBSCAN              TBSCAN
      (   7)              (   8)
      162.206             112.738
        53                  16
        |                   |
      490864                100
   CO-TABLE: COLORG    CO-TABLE: COLORG
       HIST_SHAD          BRANCH_SHAD
          Q2                  Q1


Operator Symbols :
------------------

   Symbol      Description
   ---------   ------------------------------------------
   >JOIN     : Left outer join
    JOIN<    : Right outer join
   >JOIN<    : Full outer join
   xJOIN     : Left antijoin
    JOINx    : Right antijoin
   ^JOIN     : Left early out
    JOIN^    : Right early out
    ATQ      : Asynchrony
    BTQ      : Broadcast
    CTQ      : Column-organized data
    DTQ      : Directed
    LTQ      : Intra-partition parallelism
    MTQ      : Merging (sorted)
    STQ      : Scatter
   RCTQ      : Column-organized data with row as the source
    XTQ      : XML aggregation
     TQ*     : Listener


Extended Diagnostic Information:
--------------------------------

Diagnostic Identifier:  1
Diagnostic Details:     EXP0148W  The following MQT or statistical view was
                 considered in query matching: "COLORG  ".
                 "BRANCH_SHAD".
Diagnostic Identifier:  2
Diagnostic Details:     EXP0148W  The following MQT or statistical view was
                 considered in query matching: "COLORG  ".
                 "HIST_SHAD".
Diagnostic Identifier:  3
Diagnostic Details:     EXP0149W  The following MQT was used (from those
                 considered) in query matching: "COLORG  ".
                 "BRANCH_SHAD".
Diagnostic Identifier:  4
Diagnostic Details:     EXP0149W  The following MQT was used (from those
                 considered) in query matching: "COLORG  ".
                 "HIST_SHAD".
```

The access plan shows that the column-organized shadow tables will be joined to produce the query result with an estimated cost for the sample of less than 300, which is much lower than the estimated cost with the original row-organized tables. The diagnostic information in the explain report shows that the DB2 optimizer found the two shadow table MQTs and was able to utilize them.

Now you will run a simple performance test using db2batch to execute the three SQL queries that reference the row-organized tables. In the first execution we will omit the SQL SET CURRENT statements necessary for the DB2 optimizer to route the query processing to the shadow tables. The file *testqrows.sql* contains the SQL queries and one monitoring query that summarizes the total database page references that DB2 used to produce the db2batch query results.

14. Enter the following commands in the Linux terminal session:
    - **db2 connect to blutran**
    - **db2batch -d blutran -f testqrows.sql -i complete -iso cs | tee bat_rowtabs.txt**
    - **more bat_rowtabs.txt**

    The db2batch report will contain a query result showing logical page references for this db2batch execution similar to the following:

```
SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as
total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS
TOTAL_L_READS
-------------------- -------------------- -------------------- -----
---------------
                   0                47645                 1554
49199
```

    The sample report shows that over 49,000 logical page references, mostly for data pages were needed to generate these three query results. Since only row-organized tables were used, there were no references to column-organized object pages.

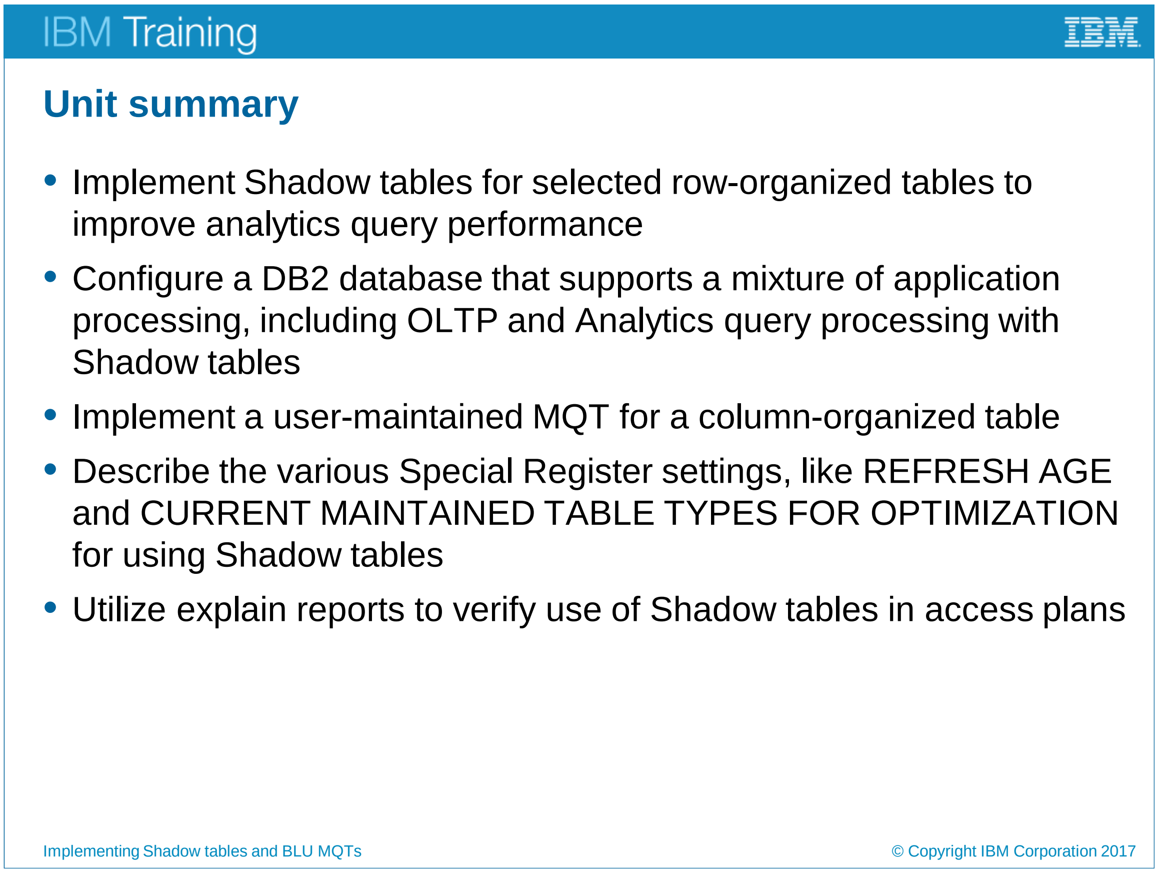
    Next you will run another db2batch to execute the same three SQL queries but the input will include the SQL SET CURRENT statements necessary for the DB2 optimizer to route the query processing to the shadow tables. The file *testqshad.sql* contains the SQL queries and one monitoring query that summarizes the total database page references that DB2 used to produce the db2batch query results.

15. Enter the following commands in the Linux terminal session:
    - **db2 connect to blutran**
    - **db2batch -d blutran -f testqshad.sql -i complete -iso cs | tee bat_shadow.txt**
    - **more bat_shadow.txt**

The db2batch report will contain a query result showing logical page references for this db2batch execution similar to the following:

```
SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as
total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS
TOTAL_L_READS
-------------------- -------------------- -------------------- -----
---------------
                 773                  311                  232
1316
```

The sample report shows that with the column-organized shadow tables being utilized, the same three SQL statements that still refer to the row-organized tables, were able to generate the query results with slightly over 1,300 total logical page references by routing the processing to the column-organized shadow tables. This is significantly less than the access required to process the row-organized tables.



IBM Training

Unit summary

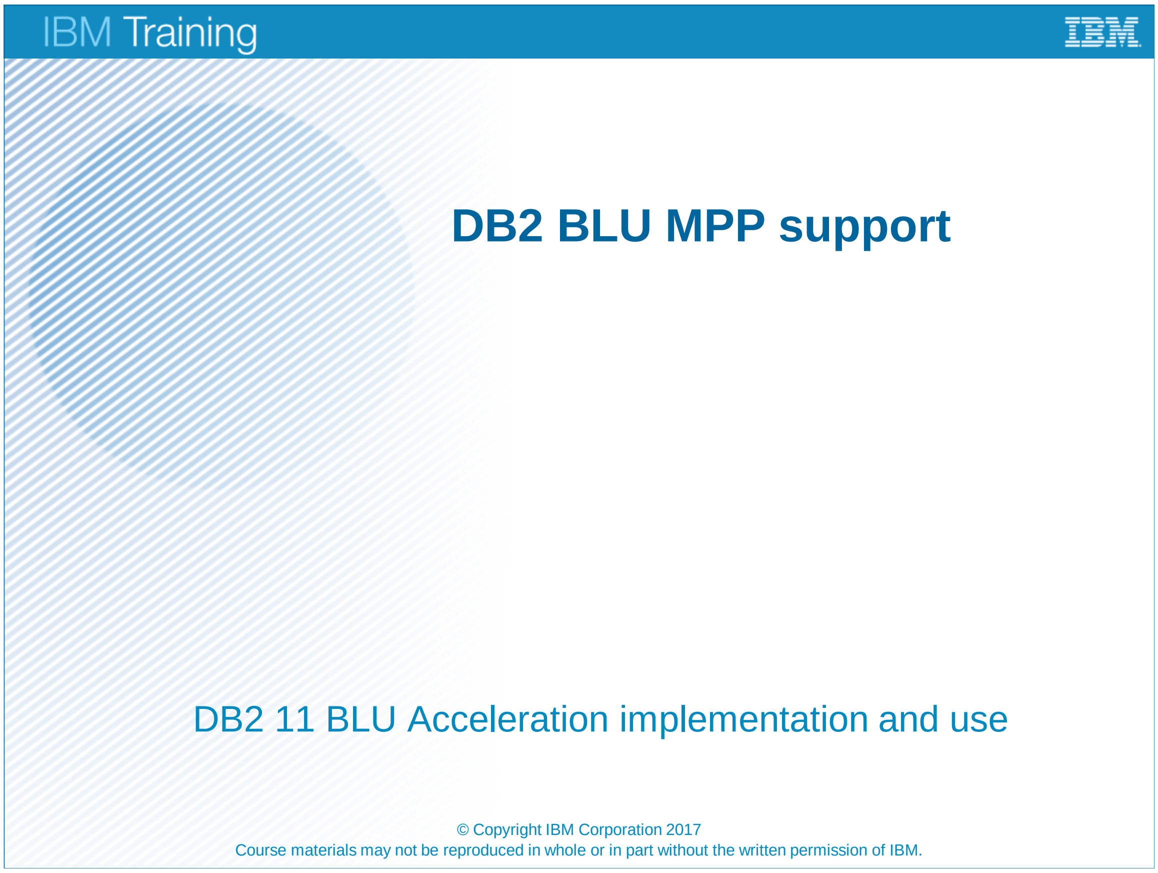
- Implement Shadow tables for selected row-organized tables to improve analytics query performance
- Configure a DB2 database that supports a mixture of application processing, including OLTP and Analytics query processing with Shadow tables
- Implement a user-maintained MQT for a column-organized table
- Describe the various Special Register settings, like REFRESH AGE and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION for using Shadow tables
- Utilize explain reports to verify use of Shadow tables in access plans

Implementing Shadow tables and BLU MQTs

© Copyright IBM Corporation 2017

Unit summary

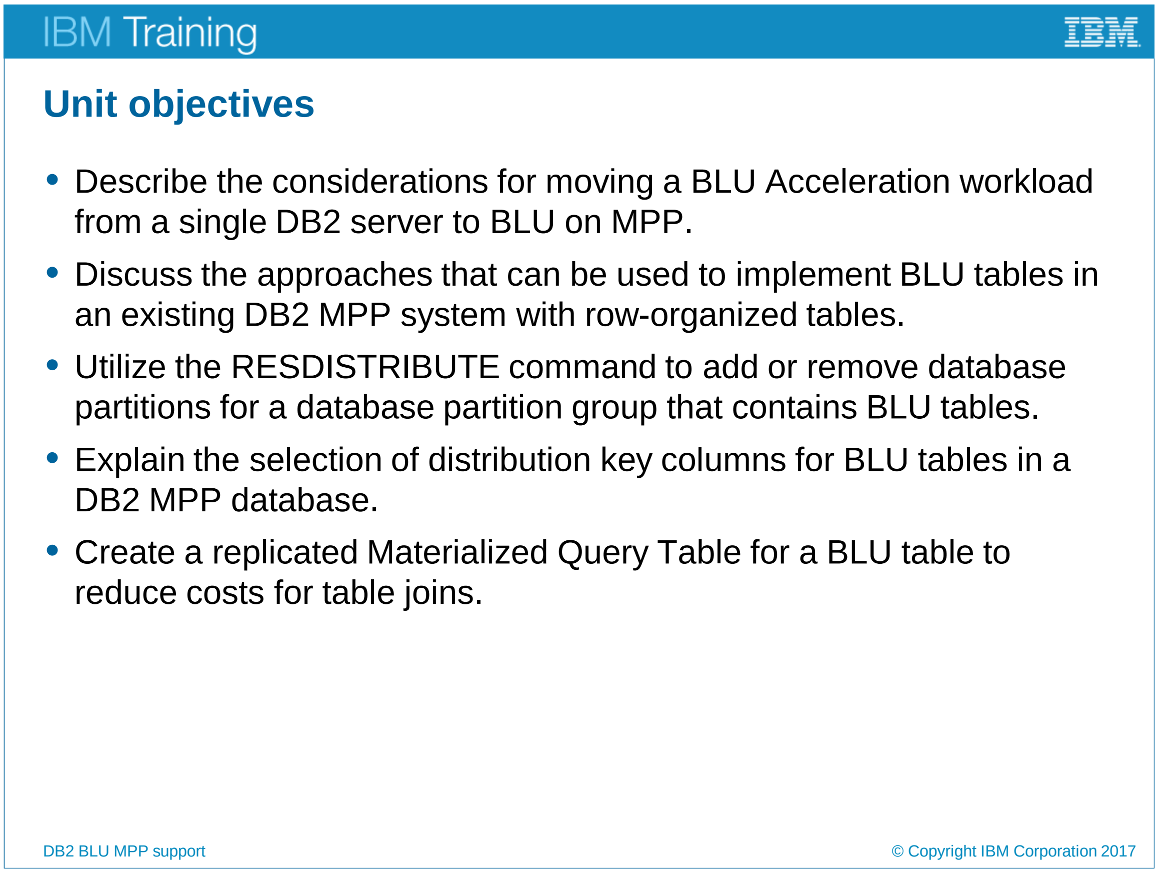
Unit 4 DB2 BLU MPP support



IBM Training

DB2 BLU MPP support

DB2 11 BLU Acceleration implementation and use



IBM Training

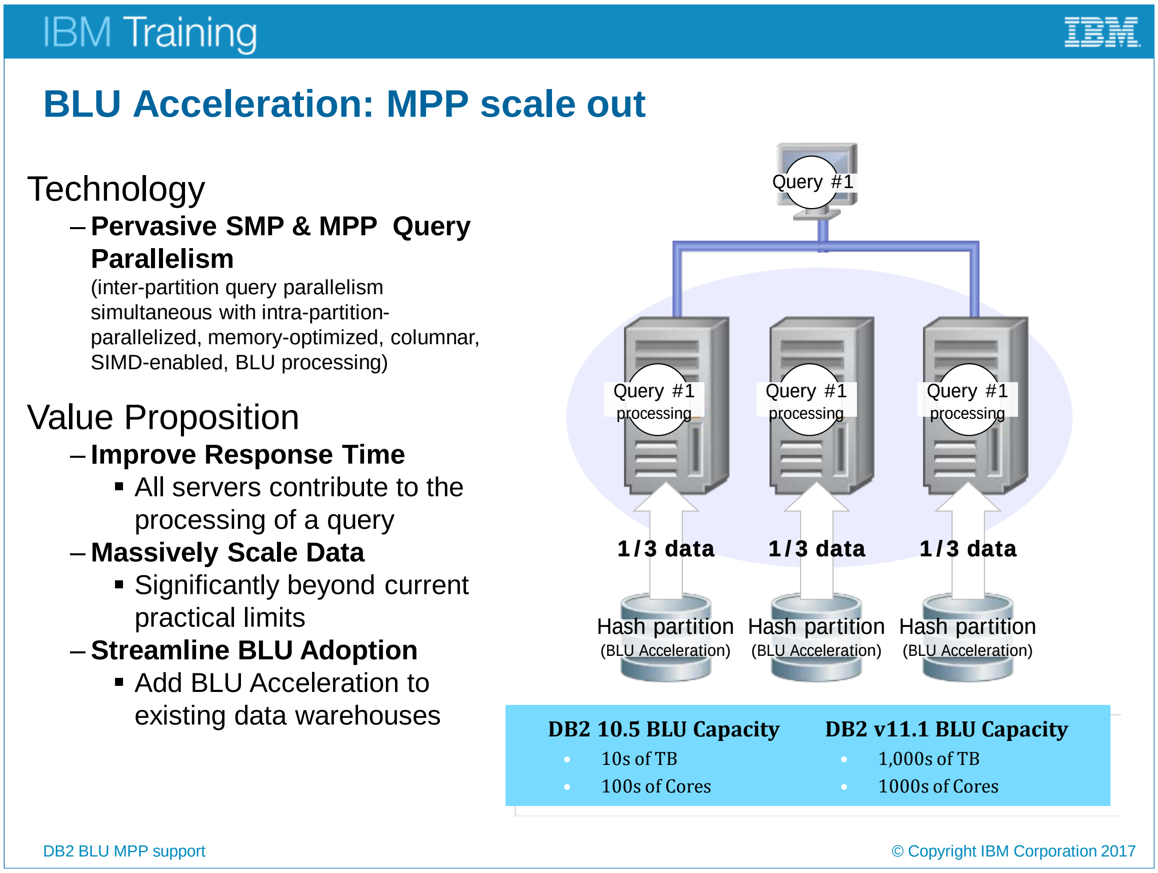
Unit objectives

- Describe the considerations for moving a BLU Acceleration workload from a single DB2 server to BLU on MPP.
- Discuss the approaches that can be used to implement BLU tables in an existing DB2 MPP system with row-organized tables.
- Utilize the RESDISTRIBUTE command to add or remove database partitions for a database partition group that contains BLU tables.
- Explain the selection of distribution key columns for BLU tables in a DB2 MPP database.
- Create a replicated Materialized Query Table for a BLU table to reduce costs for table joins.

DB2 BLU MPP support

© Copyright IBM Corporation 2017

Unit objectives



IBM Training

BLU Acceleration: MPP scale out

Technology

– **Pervasive SMP & MPP Query Parallelism**
(inter-partition query parallelism simultaneous with intra-partition-parallelized, memory-optimized, columnar, SIMD-enabled, BLU processing)

Value Proposition

– **Improve Response Time**
  - All servers contribute to the processing of a query

– **Massively Scale Data**
  - Significantly beyond current practical limits

– **Streamline BLU Adoption**
  - Add BLU Acceleration to existing data warehouses

Query #1
Query #1 processing
Query #1 processing
Query #1 processing
1 / 3 data
1 / 3 data
1 / 3 data
Hash partition (BLU Acceleration)
Hash partition (BLU Acceleration)
Hash partition (BLU Acceleration)

DB2 10.5 BLU Capacity	DB2 v11.1 BLU Capacity
• 10s of TB	• 1,000s of TB
• 100s of Cores	• 1000s of Cores

DB2 BLU MPP support

© Copyright IBM Corporation 2017

BLU Acceleration: MPP scale out

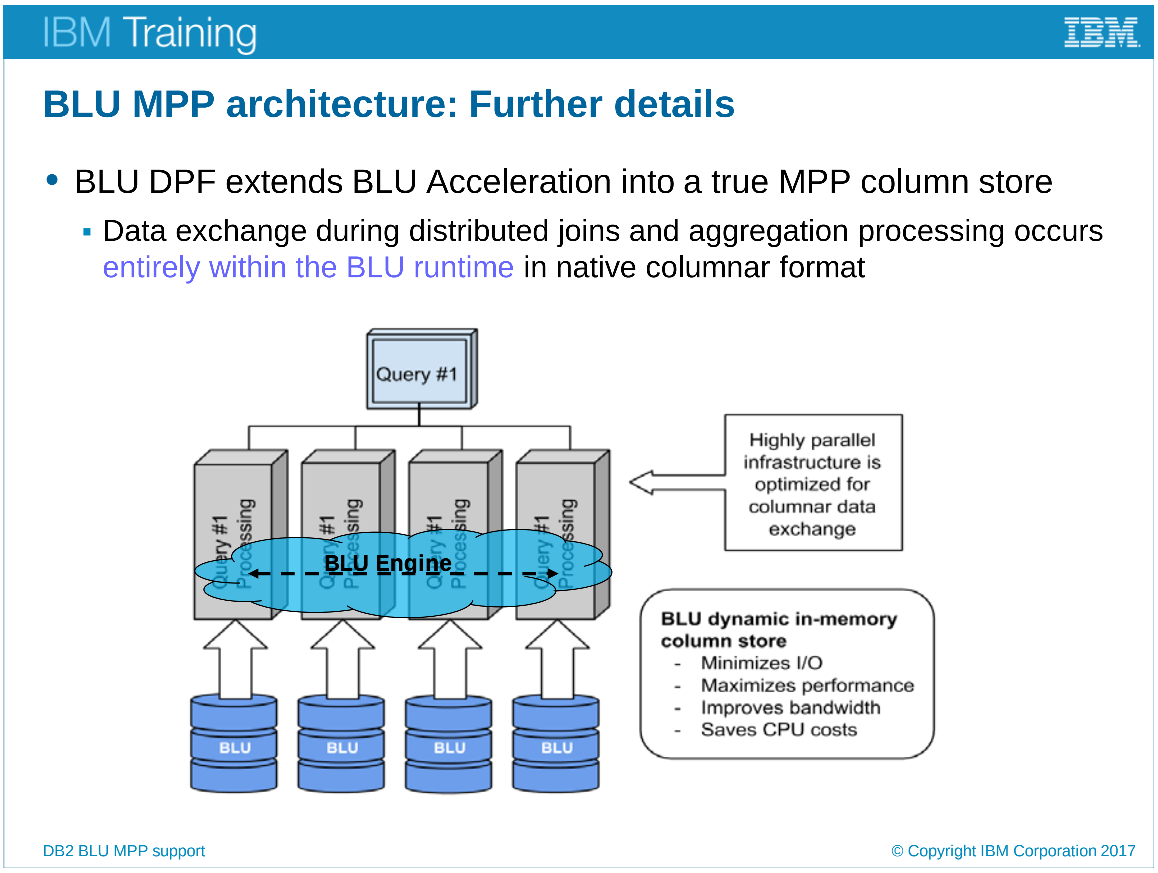
DB2 Version 11.1 extends compressed column-organized tables to partitioned DB2 database environments, allowing you to leverage BLU Acceleration, a combination of innovations from IBM Research and the development labs that simplify and speed up reporting and analytics, at massive scale, through DB2's MPP architecture. Streamlined adoption of BLU also means that existing MPP data warehouses can easily leverage the in-memory optimized columnar technology.

The following specific enhancements pertain to column-organized tables in a partitioned environment:

- MPP-aware query planning and optimization for the column vector processing engine
- an optimized vector format for columnar data exchange between partitions
- a common table dictionary allowing data to remain compressed across the network
- an optimized communications infrastructure designed for multi-core parallelism

With DB2 10.5, BLU Acceleration provided dramatic performance gains for analytics query processing, but the data size was limited to a single DB2 server.

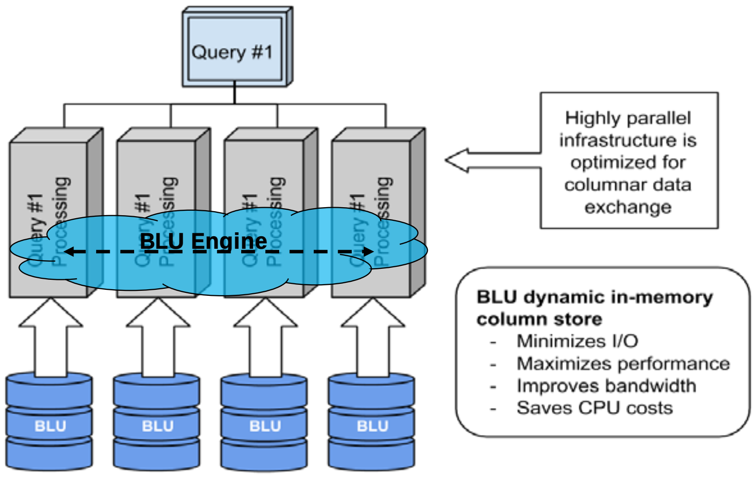
With DB2 11.1, BLU Acceleration now can be scaled to a cluster of DB2 servers, using the shared-nothing architecture of a DB2 partitioned database.



IBM Training

BLU MPP architecture: Further details

- BLU DPF extends BLU Acceleration into a true MPP column store
  - Data exchange during distributed joins and aggregation processing occurs entirely within the BLU runtime in native columnar format



Highly parallel infrastructure is optimized for columnar data exchange

BLU dynamic in-memory column store
- Minimizes I/O
- Maximizes performance
- Improves bandwidth
- Saves CPU costs

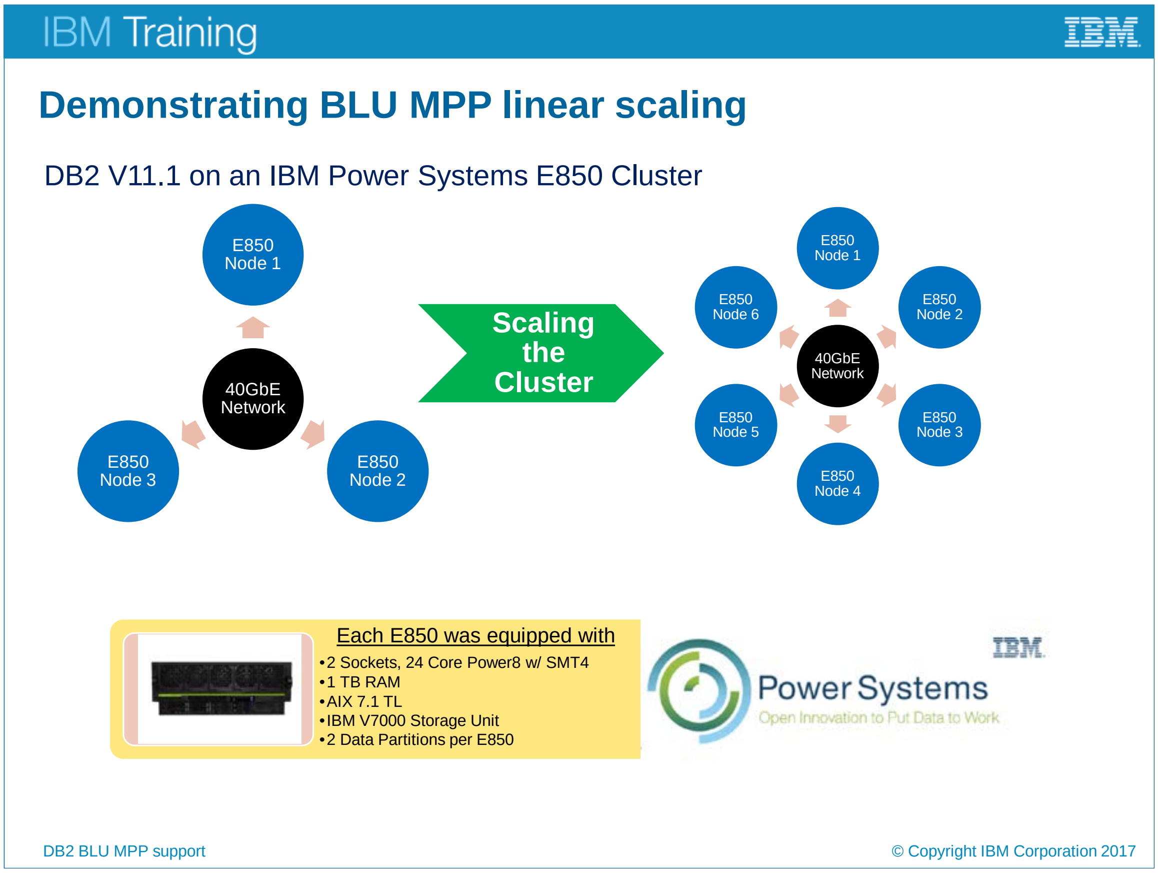
DB2 BLU MPP support

© Copyright IBM Corporation 2017

BLU MPP architecture: further details

With the BLU MPP support in DB2 11.1, each data partition leverages BLU Acceleration's dynamic in-memory column store technology using industry-leading compression, SIMD processing, and data skipping techniques.

As a query is processed, data is exchanged between nodes while keeping the data in native columnar format. There is no need to convert the data into row-organized form for this data exchange. Operations like joins and aggregations can benefit from this new technology.



IBM Training

Demonstrating BLU MPP linear scaling

DB2 V11.1 on an IBM Power Systems E850 Cluster

E850 Node 1
40GbE Network
E850 Node 3
E850 Node 2
Scaling the Cluster
E850 Node 1
E850 Node 6
E850 Node 2
40GbE Network
E850 Node 5
E850 Node 3
E850 Node 4
Power Systems
Open Innovation to Put Data to Work

Each E850 was equipped with

- 2 Sockets, 24 Core Power8 w/ SMT4
- 1 TB RAM
- AIX 7.1 TL
- IBM V7000 Storage Unit
- 2 Data Partitions per E850

DB2 BLU MPP support

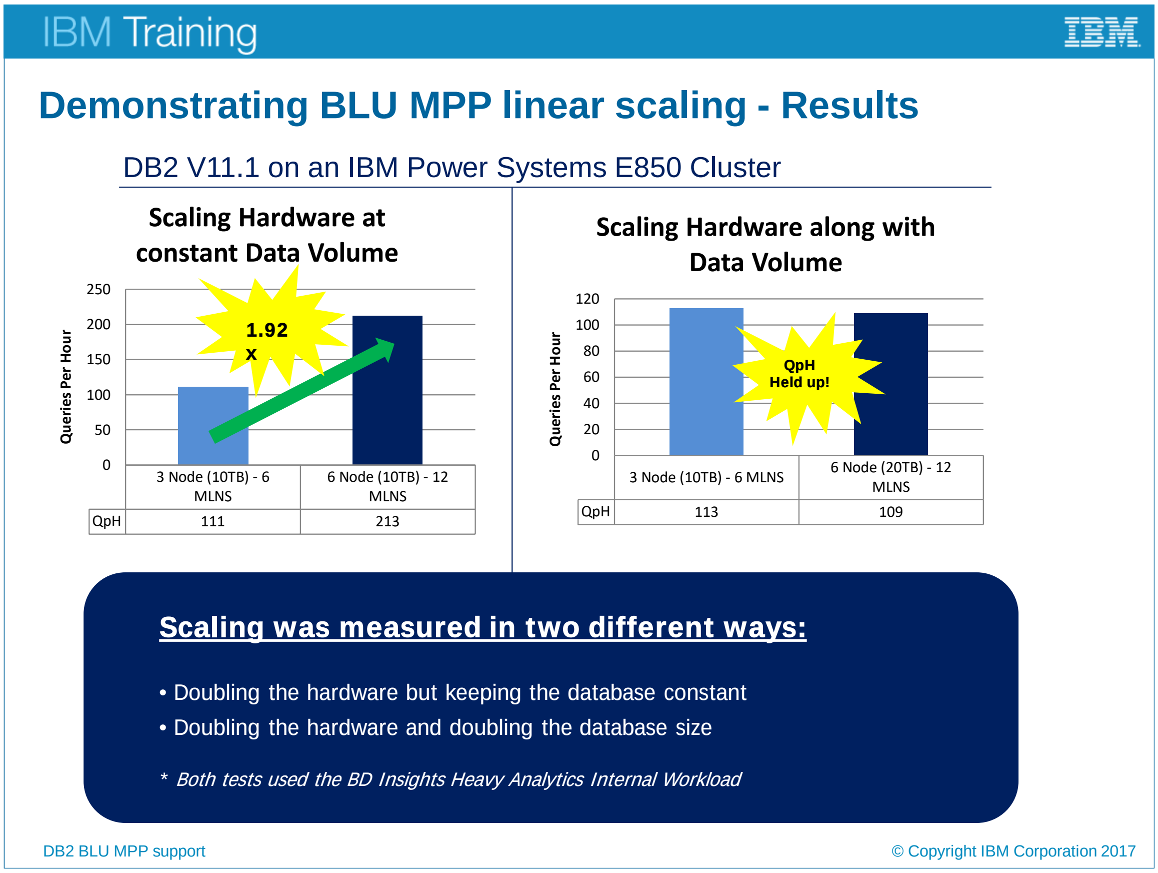
© Copyright IBM Corporation 2017

Demonstrating BLU MPP linear scaling

The slide shows two server clusters that were used to perform scaling tests.

- The first cluster contained three IBM Power E850 servers.
- The second cluster doubled the cluster size to six IBM Power E850 servers.

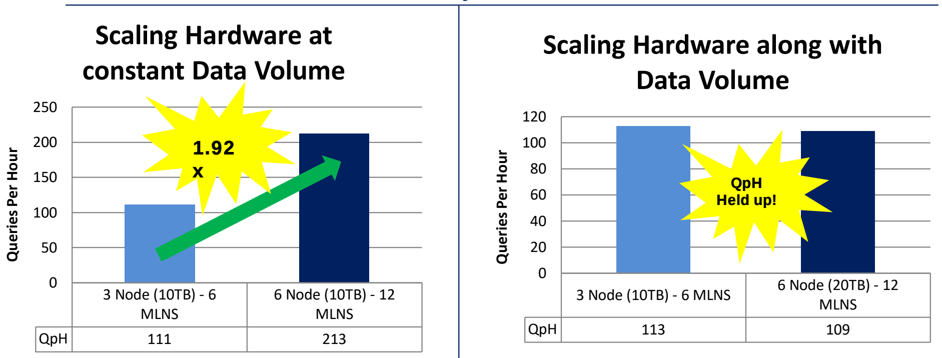
The results of the scaling test are on the next slide.



IBM Training

Demonstrating BLU MPP linear scaling - Results

DB2 V11.1 on an IBM Power Systems E850 Cluster



	3 Node (10TB) - 6 MLNS	6 Node (10TB) - 12 MLNS
QpH	111	213

	3 Node (10TB) - 6 MLNS	6 Node (20TB) - 12 MLNS
QpH	113	109

Scaling was measured in two different ways:

- Doubling the hardware but keeping the database constant
- Doubling the hardware and doubling the database size

** Both tests used the BD Insights Heavy Analytics Internal Workload*

DB2 BLU MPP support © Copyright IBM Corporation 2017

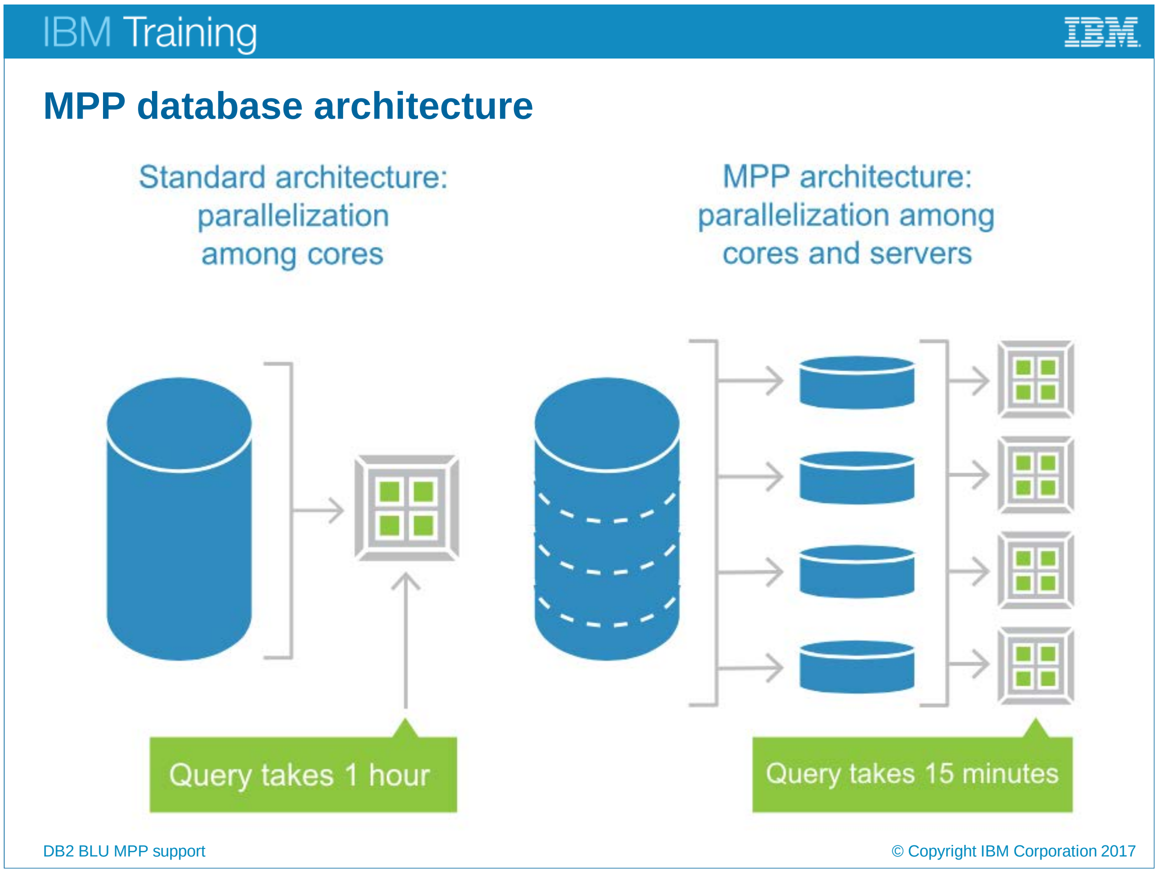
Demonstrating BLU MPP linear scaling: Results

The workload being used for the scaling test was *BD Insights*, an IBM internal workload based on the TPC-DS schema, with a multi-user apparatus through Apache JMeter.

The three node cluster had an approximately 10TB raw data set loaded into it. The six node cluster also had a 10TB database and a 20TB database as well.

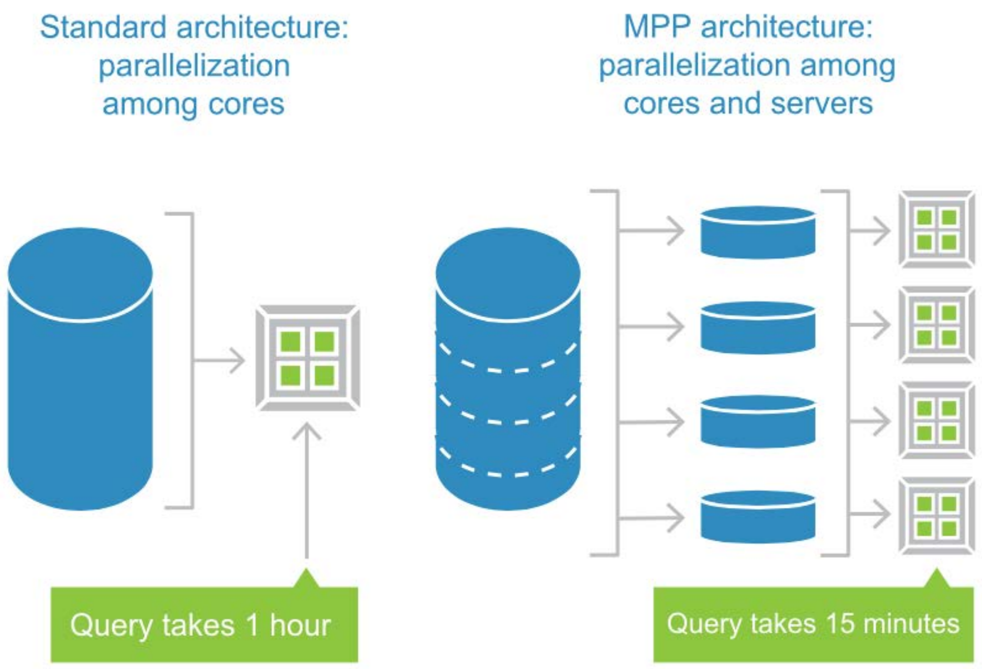
The performance test's purpose was to measure scaling on two dimensions: with constant data and scaled data:

- In the constant data scenario (three nodes versus six nodes, both using the 10 TB database), a perfect scaling story would be that the six nodes can finish the batch in half the time of the three node cluster. The result was very close to this (213 versus 111 Queries per Hour throughput score).
- In the scaling data scenario (three nodes at 10TB, compared to six nodes with 20TB), a perfect scaling story would be both sets finishing their batches in the same amount of time. Again, the test result gets very close with a small percentage difference (113 versus 109 Queries per Hour throughput score).



IBM Training

MPP database architecture


Standard architecture: parallelization among cores
MPP architecture: parallelization among cores and servers
Query takes 1 hour
Query takes 15 minutes

DB2 BLU MPP support

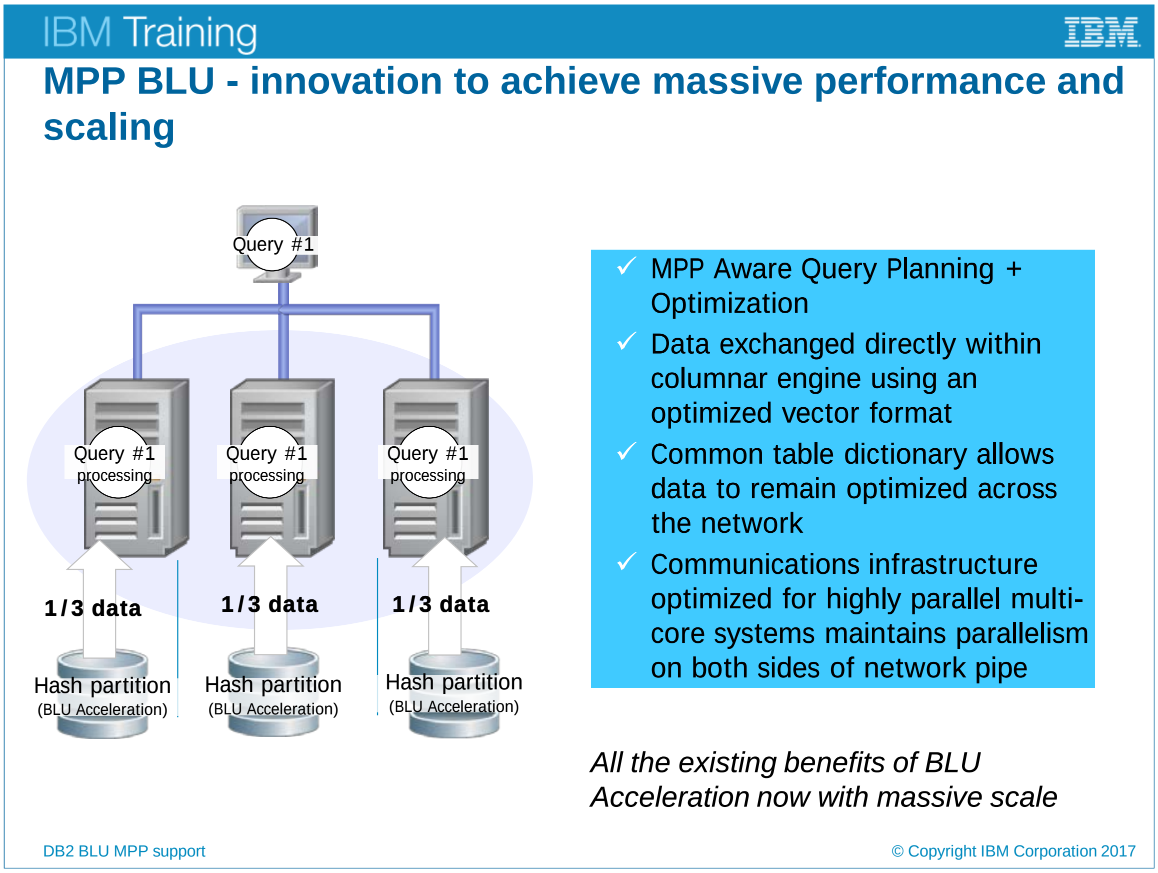
© Copyright IBM Corporation 2017

MPP database architecture

The general concept used for DB2 partitioned databases is to take the full application data that would be accessed by a single database server and evenly divide the data across multiple database servers, transparent to the application.

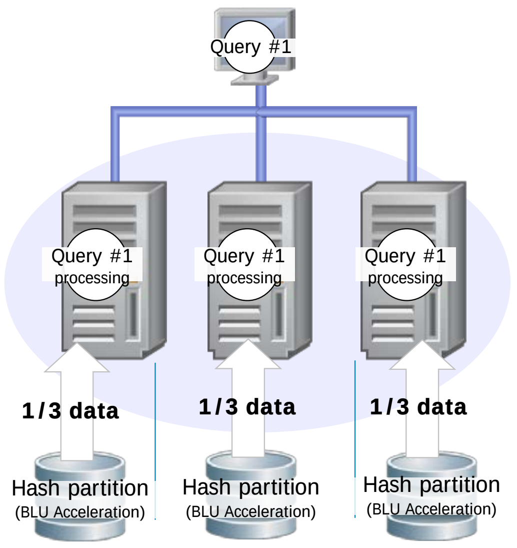
Each of the database servers can process a SQL query on its portion of the data in parallel, so the complete result can be produced much faster.

For example, a query that takes one hour to complete on a single server may be able to produce the same result in fifteen minutes, if 25 percent of the data is stored on four servers.



IBM Training

MPP BLU - innovation to achieve massive performance and scaling


Query #1
Query #1 processing
Query #1 processing
Query #1 processing
1 / 3 data
1 / 3 data
1 / 3 data
Hash partition (BLU Acceleration)
Hash partition (BLU Acceleration)
Hash partition (BLU Acceleration)

- ✓ MPP Aware Query Planning + Optimization
- ✓ Data exchanged directly within columnar engine using an optimized vector format
- ✓ Common table dictionary allows data to remain optimized across the network
- ✓ Communications infrastructure optimized for highly parallel multi-core systems maintains parallelism on both sides of network pipe

All the existing benefits of BLU Acceleration now with massive scale

DB2 BLU MPP support

© Copyright IBM Corporation 2017

MPP BLU: innovation to achieve massive performance and scaling

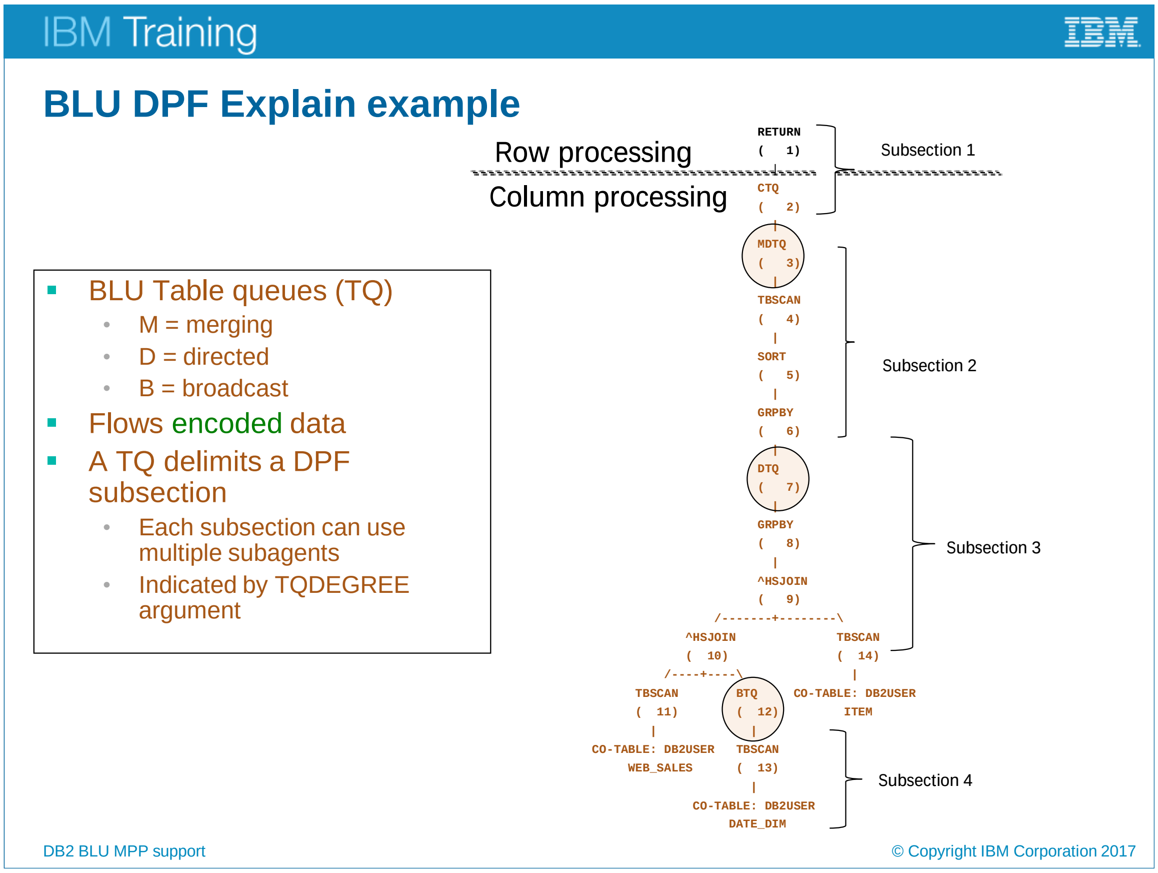
DB2 11.1 now provides the performance benefits of column-organized BLU table support to DB2 MPP databases to enable massive scaling potential.

The DB2 optimizer in DB2 11.1 was enhanced to support access to BLU tables with awareness of the additional considerations for execution in a MPP database.

With DB2 11.1 on a MPP database, data can be passed using table queues between database partitions within the columnar data engine, so it can be processed in optimized vector format.

The column dictionary data for a table is copied to every database partition, so that data can move between database partitions without needing to decode it.

The inter-partition communication component, the Fast Communications Manager (FCM), allows a high degree of parallel processing, even when data is moved between partitions.



IBM Training

BLU DPF Explain example

- BLU Table queues (TQ)
  - M = merging
  - D = directed
  - B = broadcast
- Flows encoded data
- A TQ delimits a DPF subsection
  - Each subsection can use multiple subagents
  - Indicated by TQDEGREE argument

Row processing

Column processing

```
                      RETURN
                      (   1)      Subsection 1
                        |
                      CTQ
                      (   2)
                        |
                      MDTQ
                      (   3)
                        |
                      TBSCAN
                      (   4)
                        |
                      SORT        Subsection 2
                      (   5)
                        |
                      GRPBY
                      (   6)
                        |
                      DTQ
                      (   7)
                        |
                      GRPBY
                      (   8)
                        |
                      ^HSJOIN         Subsection 3
                      (   9)
                  /-------+--------\
             ^HSJOIN              TBSCAN
             (  10)               (  14)
          /----+----\               |
      TBSCAN        BTQ      CO-TABLE: DB2USER
      (  11)      (  12)          ITEM
        |            |
CO-TABLE: DB2USER  TBSCAN
    WEB_SALES      (  13)      Subsection 4
                     |
           CO-TABLE: DB2USER
               DATE_DIM
```

DB2 BLU MPP support

© Copyright IBM Corporation 2017

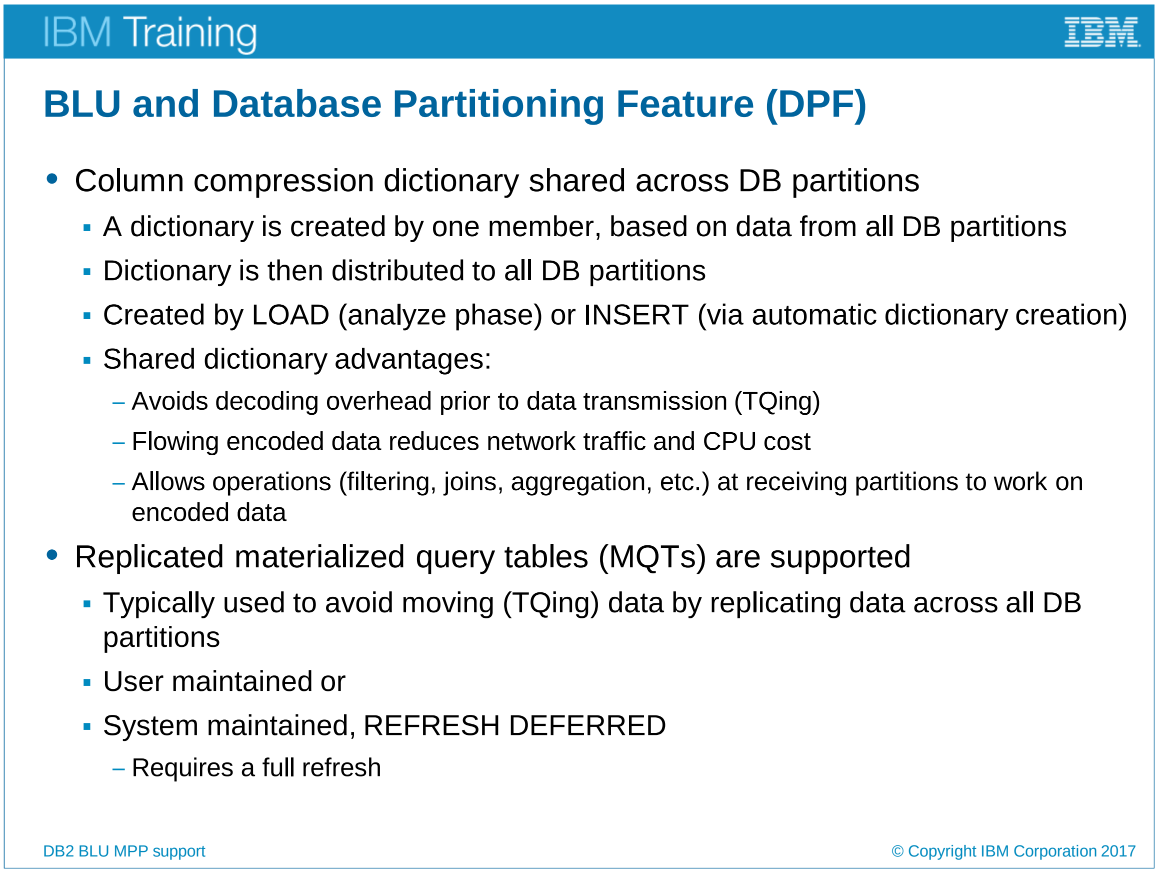
BLU DPF Explain example

The slide shows a sample access plan, joining three column-organized tables. The access plan contains four subsections, joined by three table queues.

The table queues are:

- A broadcast table queue (BTQ), step 12, links subsections 4 and 3.
- A directed table queue (DTQ), step 7, links subsections 3 and 2.
- A merged directed table queue (MDTQ), step 3, links subsections 2 and 1.

All of these are performed below the CTQ, step 2, so they are executed using the columnar data engine.



IBM Training

BLU and Database Partitioning Feature (DPF)

- Column compression dictionary shared across DB partitions
  - A dictionary is created by one member, based on data from all DB partitions
  - Dictionary is then distributed to all DB partitions
  - Created by LOAD (analyze phase) or INSERT (via automatic dictionary creation)
  - Shared dictionary advantages:
    - Avoids decoding overhead prior to data transmission (TQing)
    - Flowing encoded data reduces network traffic and CPU cost
    - Allows operations (filtering, joins, aggregation, etc.) at receiving partitions to work on encoded data
- Replicated materialized query tables (MQTs) are supported
  - Typically used to avoid moving (TQing) data by replicating data across all DB partitions
  - User maintained or
  - System maintained, REFRESH DEFERRED
    - Requires a full refresh

DB2 BLU MPP support

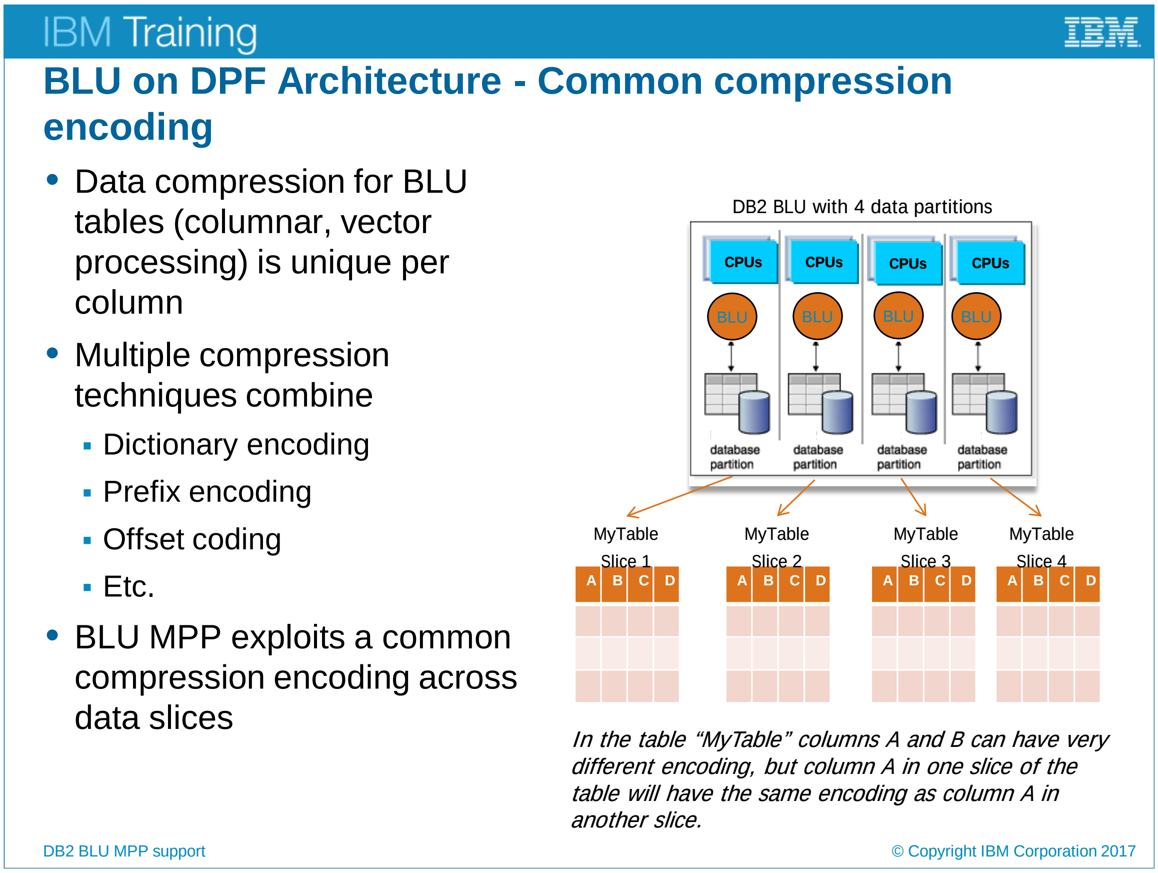
BLU and Database Partitioning Feature (DPF)

In a partitioned database, each data row resides on one database partition. Since BLU tables are all compressed using column dictionaries, a column dictionary local to the data would only be able to decode data on that one partition.

The column compression dictionaries for BLU tables on MPP databases are built by sending the local column dictionary data to create a global column dictionary for the table that is distributed to all database partitions.

With a global shared column dictionary, encoded data for a column can move between partitions without being decoded. It also allows the encoded column data to be processed in arrays on any database partition.

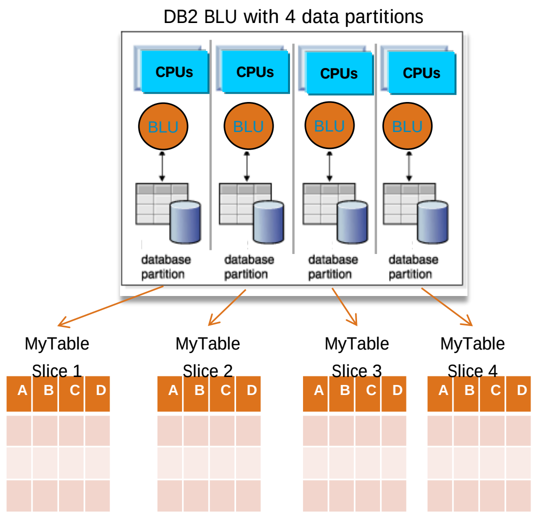
With DB2 11.1, you can create replicated materialized query tables that are column-organized. This may be done to allow collocated joins to be performed.



IBM Training

BLU on DPF Architecture - Common compression encoding

- Data compression for BLU tables (columnar, vector processing) is unique per column
- Multiple compression techniques combine
  - Dictionary encoding
  - Prefix encoding
  - Offset coding
  - Etc.
- BLU MPP exploits a common compression encoding across data slices



In the table “MyTable” columns A and B can have very different encoding, but column A in one slice of the table will have the same encoding as column A in another slice.

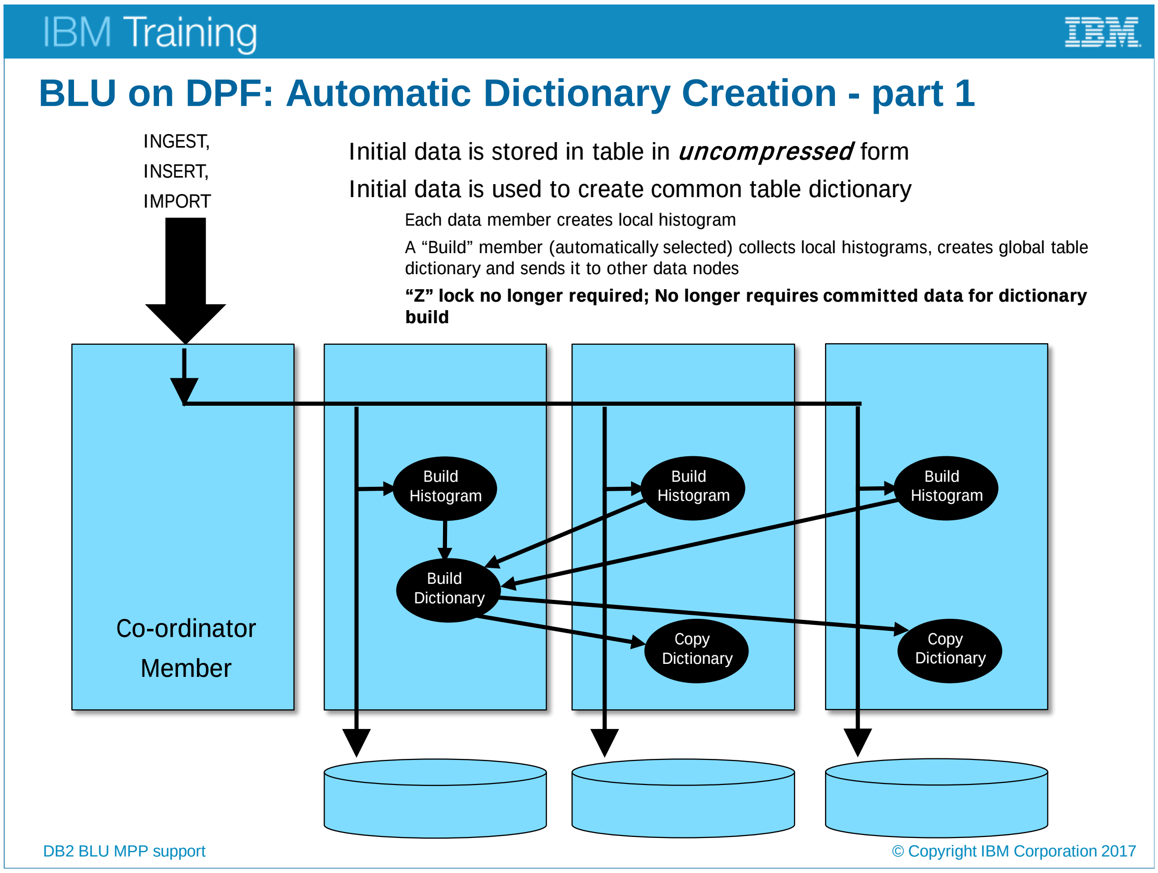
DB2 BLU MPP support © Copyright IBM Corporation 2017

BLU on DPF architecture: Common compression encoding

DB2 MPP databases have a shared-nothing concept, so objects like table indexes are local to the data at a partition level rather than being global.

For the column dictionary data associated with DB2 BLU tables, DB2 11.1 implements a common global column dictionary that spans all database partitions. Having a common column dictionary for each column ensures that a value in a column will be compressed into the same encoded form for every partition.

This makes it possible to send encoded column data from one partition to another without first decoding the data. This reduces the communications cost and time for moving encoded data between partitions.


IBM Training

BLU on DPF: Automatic Dictionary Creation - part 1

- Initial data is stored in table in ***uncompressed*** form
- Initial data is used to create common table dictionary
  - Each data member creates local histogram
  - A "Build" member (automatically selected) collects local histograms, creates global table dictionary and sends it to other data nodes
  - **"Z" lock no longer required; No longer requires committed data for dictionary build**

DB2 BLU MPP support © Copyright IBM Corporation 2017

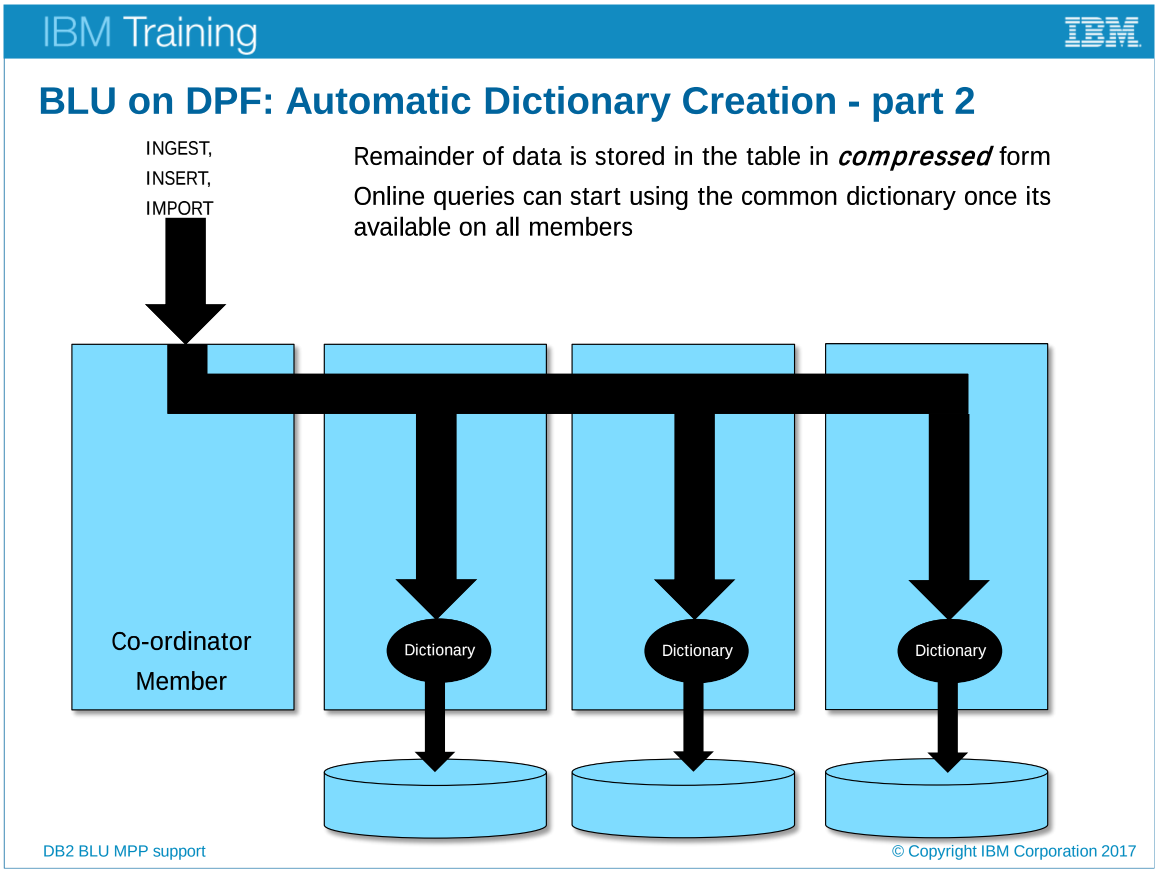
BLU on DPF: Automatic Dictionary Creation - part 1

The LOAD utility utilizes the ANALYZE phase to scan the load input to build the column compression dictionaries for BLU tables.

The INGEST and IMPORT utilities and applications using SQL INSERT statements to load data into tables utilize the automatic dictionary creation routines in DB2 to build the column compression dictionary when data is inserted into a new BU table.

With BLU MPP, the automatic dictionary creation function requires several steps:

1. Data being inserted into the new table arrives at each database partition and is added to the BLU table in an uncompressed or unencoded form until a size threshold is reached.
2. DB2 uses that initial data to build histograms for each column based on the local partition data.
3. One partition combines the histograms from all partitions into a set of common column dictionaries.
4. The completed common column dictionary is copied to all database partitions where the table has storage. This copying is based on the database partition group of the table space holding the table.



IBM Training

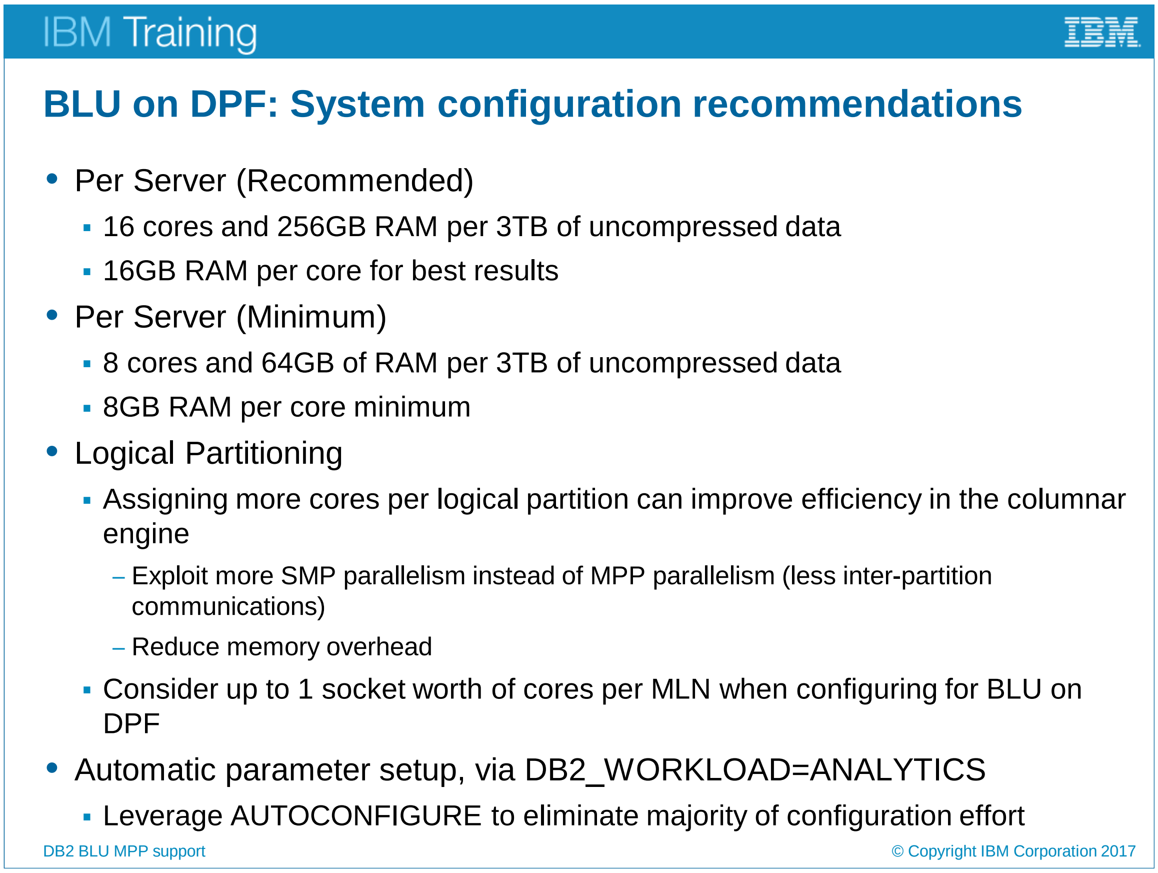
BLU on DPF: Automatic Dictionary Creation - part 2

- Remainder of data is stored in the table in ***compressed*** form
- Online queries can start using the common dictionary once its available on all members

BLU on DPF: Automatic Dictionary Creation - part 2

Once the common column dictionary data is available on all partitions of the BLU table, new data being inserted by the application or DB2 utility can use the dictionary data to encode and compress the data.

Since the table may be available concurrently to other applications performing query processing concurrently with the new data being added, the common column dictionary data will be used to process those queries.


IBM Training

BLU on DPF: System configuration recommendations

- Per Server (Recommended)
  - 16 cores and 256GB RAM per 3TB of uncompressed data
  - 16GB RAM per core for best results
- Per Server (Minimum)
  - 8 cores and 64GB of RAM per 3TB of uncompressed data
  - 8GB RAM per core minimum
- Logical Partitioning
  - Assigning more cores per logical partition can improve efficiency in the columnar engine
    - Exploit more SMP parallelism instead of MPP parallelism (less inter-partition communications)
    - Reduce memory overhead
  - Consider up to 1 socket worth of cores per MLN when configuring for BLU on DPF
- Automatic parameter setup, via DB2_WORKLOAD=ANALYTICS
  - Leverage AUTOCONFIGURE to eliminate majority of configuration effort

DB2 BLU MPP support © Copyright IBM Corporation 2017

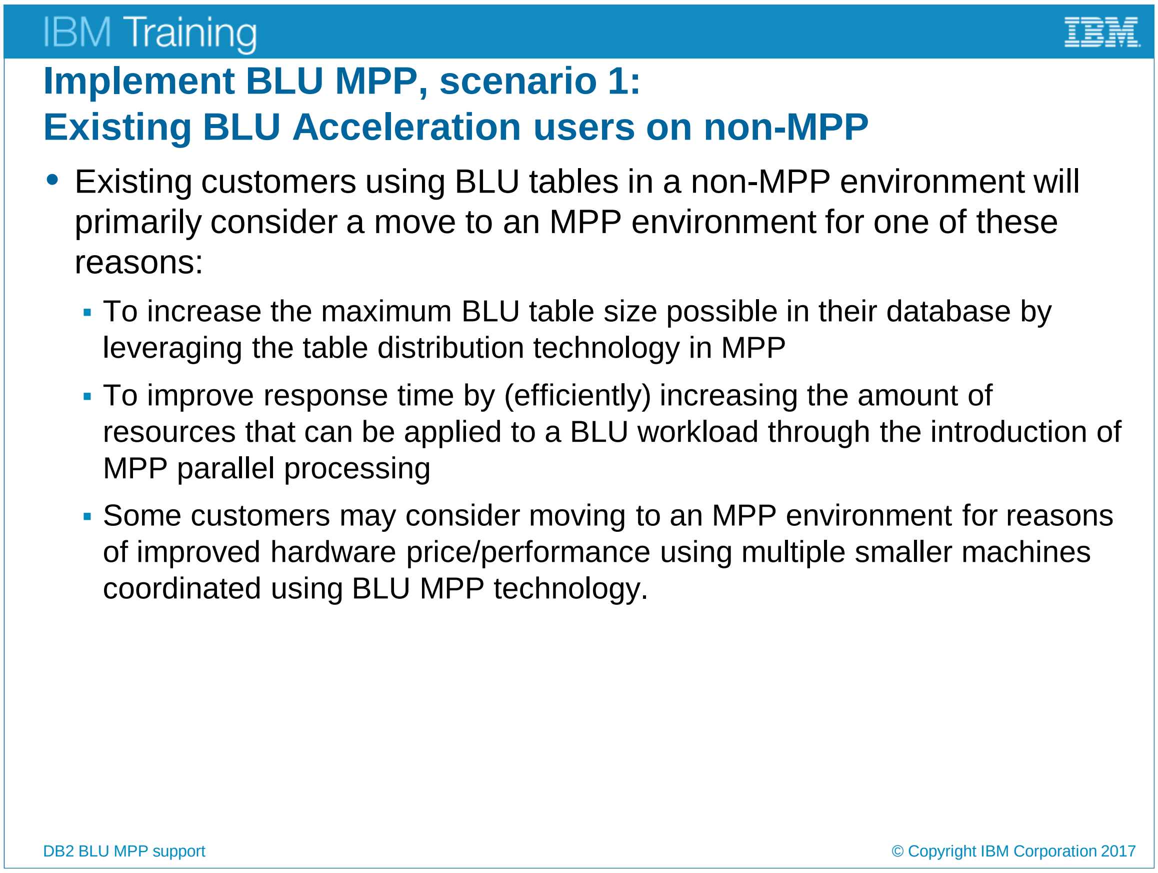
BLU on DPF: System configuration recommendations

The philosophy behind DB2 BLU is to leverage machine resources and hardware optimizations as much as possible to achieve orders of magnitude performance improvements over conventional workloads.

In order to unlock the full potential of the BLU technology, it's recommended to run it on systems that meet at least a minimum level in terms of CPU cores and system memory.

The more memory and parallelism it can exploit, the better the results will be:

- More of the data set can fit in memory, meaning less I/O
- More of the processing can fit in memory, meaning less need to land intermediate results on disk (for example, spilling of data during sort processing)
- Processing can be more heavily parallelized



IBM Training

Implement BLU MPP, scenario 1: Existing BLU Acceleration users on non-MPP

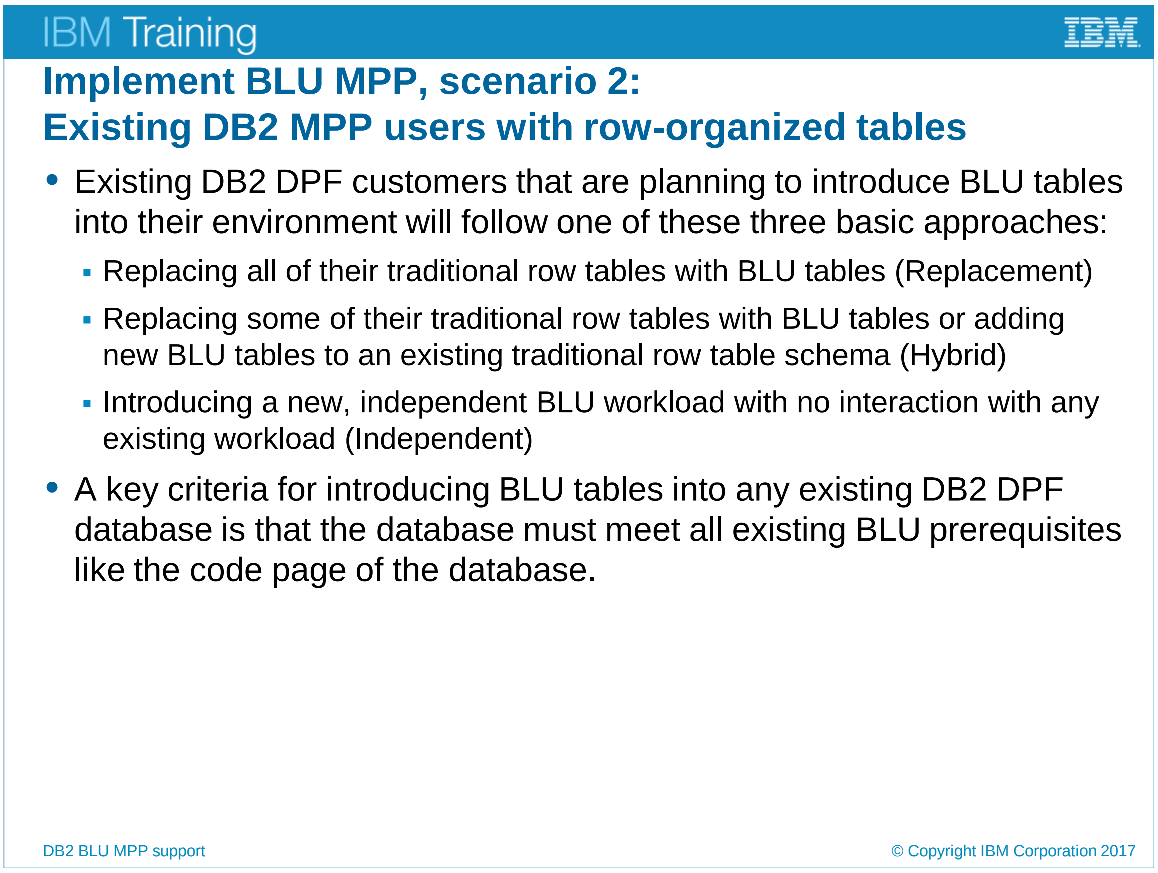
- Existing customers using BLU tables in a non-MPP environment will primarily consider a move to an MPP environment for one of these reasons:
  - To increase the maximum BLU table size possible in their database by leveraging the table distribution technology in MPP
  - To improve response time by (efficiently) increasing the amount of resources that can be applied to a BLU workload through the introduction of MPP parallel processing
  - Some customers may consider moving to an MPP environment for reasons of improved hardware price/performance using multiple smaller machines coordinated using BLU MPP technology.

DB2 BLU MPP support © Copyright IBM Corporation 2017

Implement BLU MPP, scenario 1: Existing BLU Acceleration users on non-MPP

Customers that are currently using BLU tables with version 10.5 on a single database server may consider moving the database to a MPP cluster for one of the following reasons:

- To support very large tables. The size limitations for tables in a single partition database are applied to each database partition of a MPP database.
- To improve query response times by spreading the data to multiple servers, each with processor and memory resources to perform parallel processing.
- In some cases, a customer may decide to move a database to a clustered MPP environment to handle the capacity requirements using multiple smaller hardware systems, which may reduce costs.



IBM Training

Implement BLU MPP, scenario 2: Existing DB2 MPP users with row-organized tables

- Existing DB2 DPF customers that are planning to introduce BLU tables into their environment will follow one of these three basic approaches:
  - Replacing all of their traditional row tables with BLU tables (Replacement)
  - Replacing some of their traditional row tables with BLU tables or adding new BLU tables to an existing traditional row table schema (Hybrid)
  - Introducing a new, independent BLU workload with no interaction with any existing workload (Independent)
- A key criteria for introducing BLU tables into any existing DB2 DPF database is that the database must meet all existing BLU prerequisites like the code page of the database.

DB2 BLU MPP support © Copyright IBM Corporation 2017

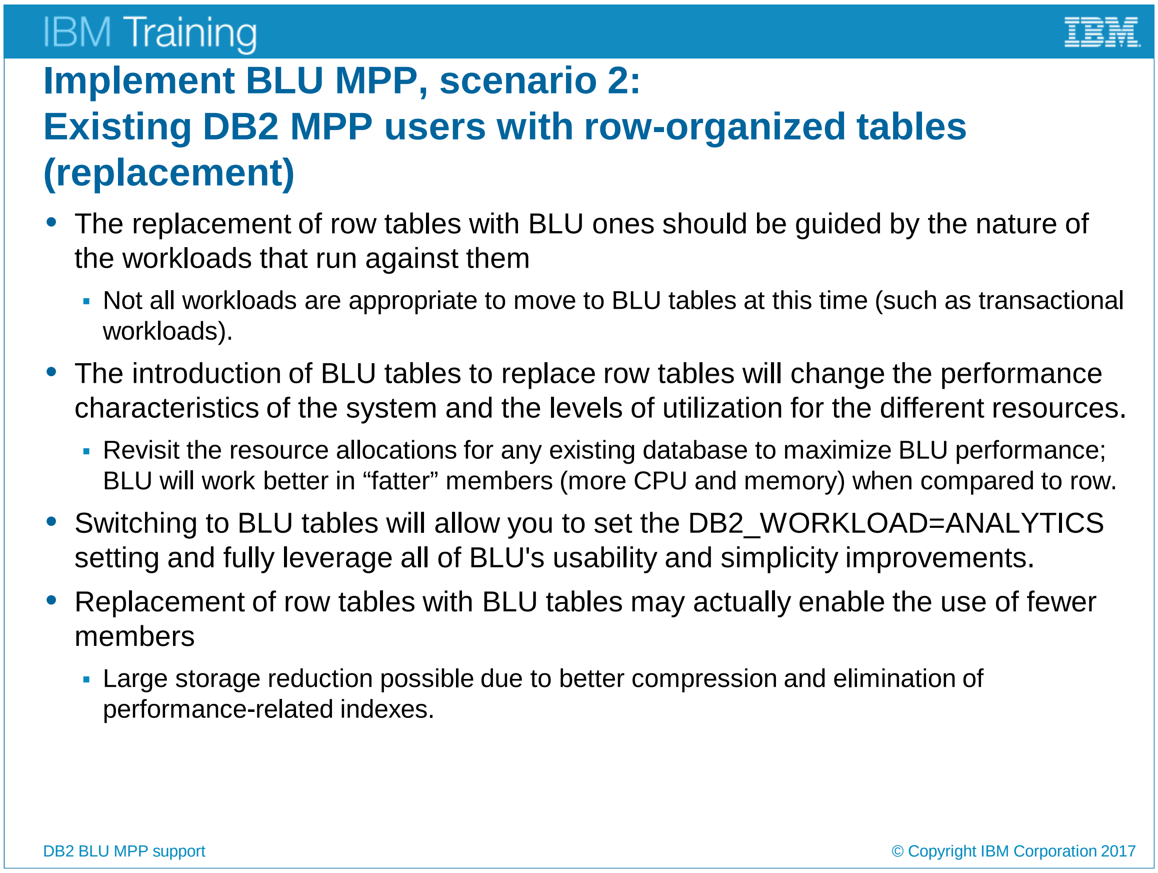
Implement BLU MPP, scenario 2: Existing DB2 MPP users with row-organized tables

There are a large number of DB2 LUW customers that have existing row-organized based applications using partitioned, MPP DB2 server clusters.

With the support for BLU tables in MPP databases, these customers can use three approaches to implement BLU tables in those databases.

- **Replacement**: some may decide to convert all of the existing row-organized tables to column-organized, BLU tables.
- **Hybrid**: some may convert selected row-organized tables to column-organized tables, or add some new column-organized tables.
- **Independent** - some may decide to leave existing applications using row-organized tables, but add new applications that make use of column-organized tables.

Note: Adding column-organized tables to an existing DB2 MPP database can only be done if the database meets all of the prerequisites for BLU tables, like a supported database code page.



IBM Training

Implement BLU MPP, scenario 2: Existing DB2 MPP users with row-organized tables (replacement)

- The replacement of row tables with BLU ones should be guided by the nature of the workloads that run against them
  - Not all workloads are appropriate to move to BLU tables at this time (such as transactional workloads).
- The introduction of BLU tables to replace row tables will change the performance characteristics of the system and the levels of utilization for the different resources.
  - Revisit the resource allocations for any existing database to maximize BLU performance; BLU will work better in “fatter” members (more CPU and memory) when compared to row.
- Switching to BLU tables will allow you to set the DB2_WORKLOAD=ANALYTICS setting and fully leverage all of BLU's usability and simplicity improvements.
- Replacement of row tables with BLU tables may actually enable the use of fewer members
  - Large storage reduction possible due to better compression and elimination of performance-related indexes.

DB2 BLU MPP support © Copyright IBM Corporation 2017

Implement BLU MPP, scenario 2: Existing DB2 MPP users with row-organized tables (replacement)

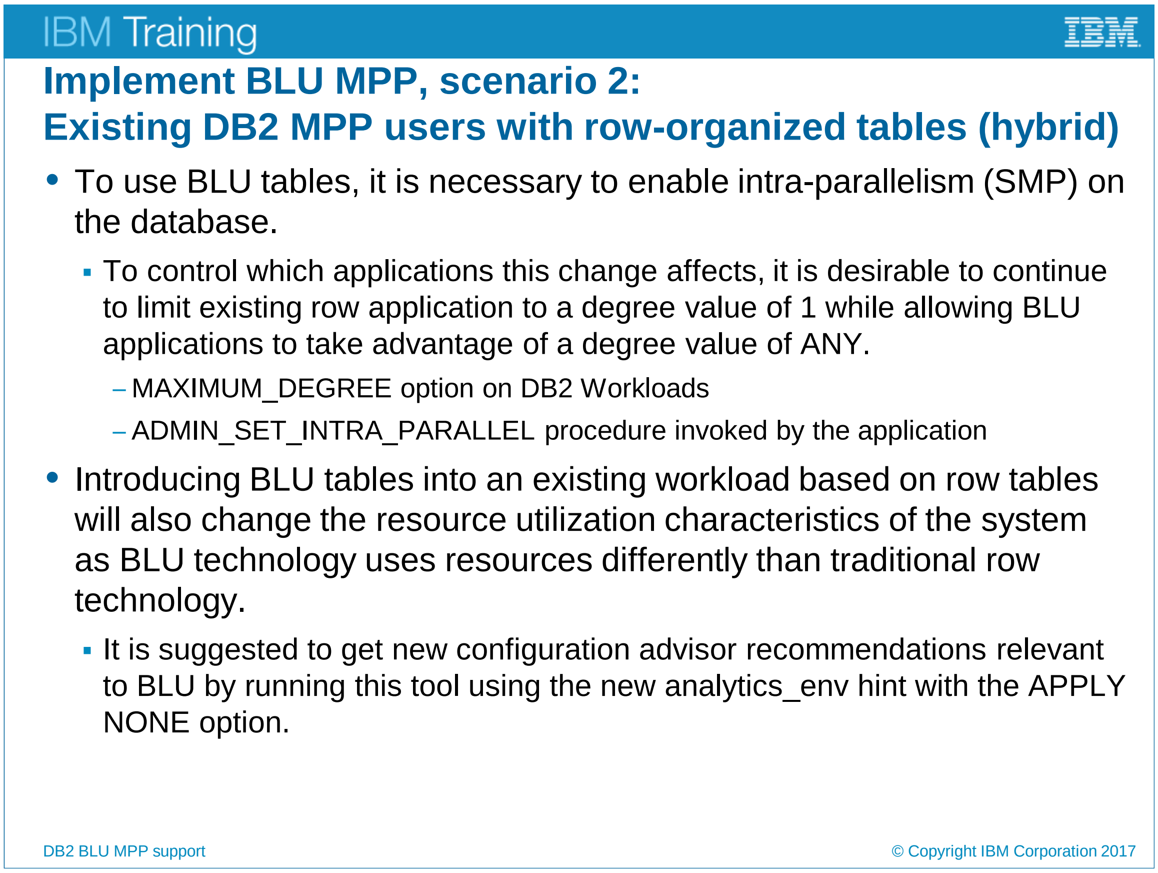
The replacement approach would be to replace **all** row-organized tables with column-organized tables.

It would be necessary to review the nature of the applications to make sure BLU tables will function efficiently for the workload. In general, you would not consider BLU tables to support a transactional workload.

Using BLU tables in a DB2 MPP database will impact the resource utilization compared to row-organized tables. DB2 BLU table processing is designed for parallel processing with large numbers of CPU processors and large memory allocations.

One key benefit of BLU tables is the simplified administration of a load and go approach to adding new data tables.

It is possible that converting row-organized tables to column-organized tables could reduce disk storage requirements and be able to handle the analytics query workload using fewer DB2 partitioned database servers.


IBM Training

Implement BLU MPP, scenario 2: Existing DB2 MPP users with row-organized tables (hybrid)

- To use BLU tables, it is necessary to enable intra-parallelism (SMP) on the database.
  - To control which applications this change affects, it is desirable to continue to limit existing row application to a degree value of 1 while allowing BLU applications to take advantage of a degree value of ANY.
    - MAXIMUM_DEGREE option on DB2 Workloads
    - ADMIN_SET_INTRA_PARALLEL procedure invoked by the application
- Introducing BLU tables into an existing workload based on row tables will also change the resource utilization characteristics of the system as BLU technology uses resources differently than traditional row technology.
  - It is suggested to get new configuration advisor recommendations relevant to BLU by running this tool using the new analytics_env hint with the APPLY NONE option.

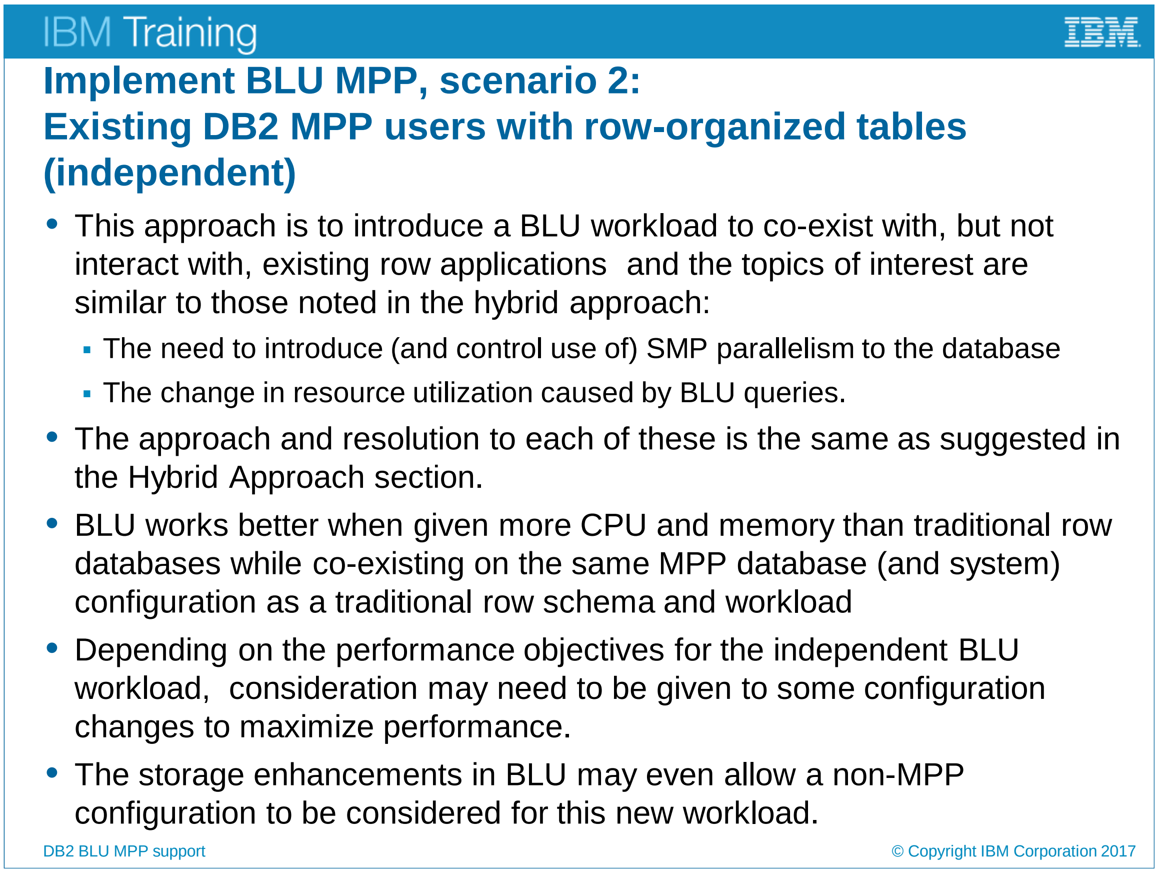
DB2 BLU MPP support © Copyright IBM Corporation 2017

Implement BLU MPP, scenario 2: Existing DB2 MPP users with row-organized tables (hybrid)

The hybrid approach uses a mixture of row-organized and column-organized tables.

BLU table support may require some database configuration adjustments, like enabling intra-parallel processing. There are several ways to enable intra-parallel processing, including setting MAXIMUM_DEGREE for a DB2 workload or using the procedure ADMIN_SET_INTRA_PARALLEL in an application.

The AUTOCONFIGURE command now has a hint, *analytics_env* that could be set and used with the APPLY NONE option to get a set of suggested configuration changes to support BLU tables in an existing database.



IBM Training

Implement BLU MPP, scenario 2: Existing DB2 MPP users with row-organized tables (independent)

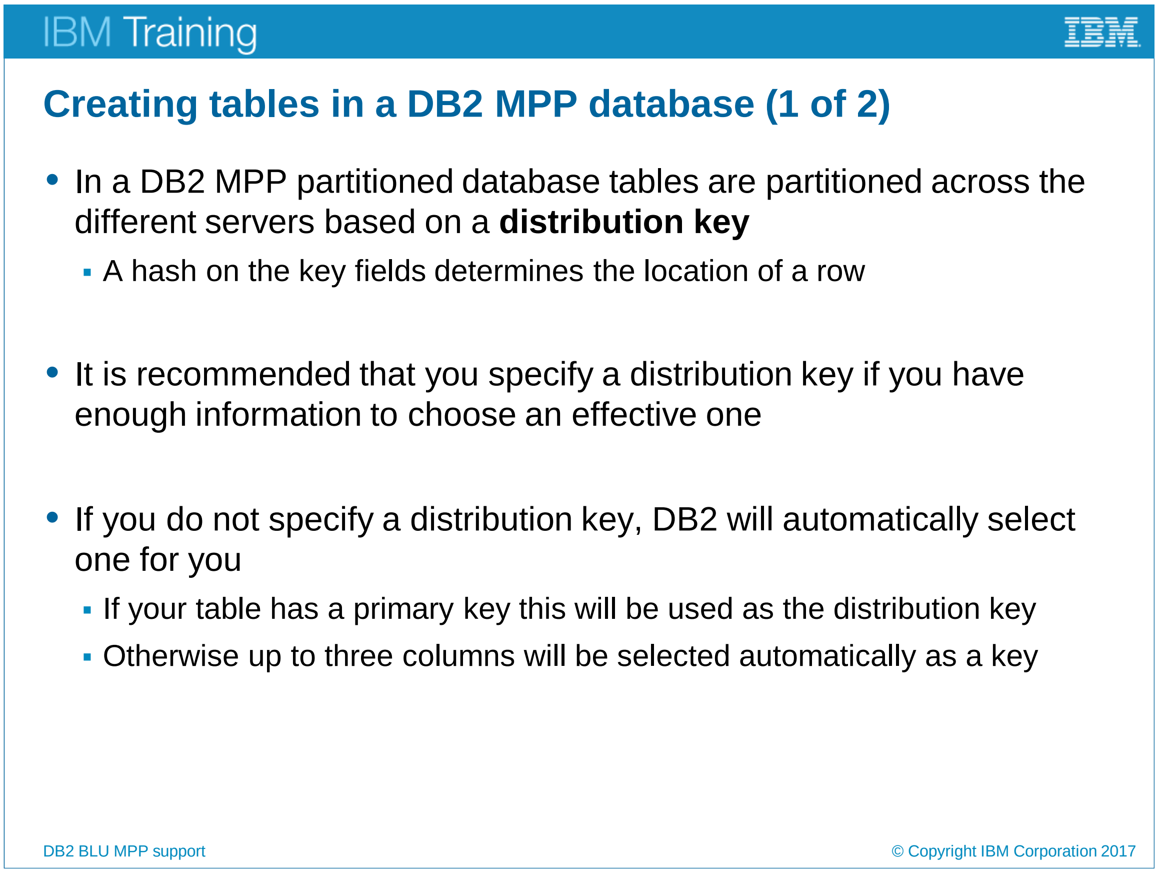
- This approach is to introduce a BLU workload to co-exist with, but not interact with, existing row applications and the topics of interest are similar to those noted in the hybrid approach:
  - The need to introduce (and control use of) SMP parallelism to the database
  - The change in resource utilization caused by BLU queries.
- The approach and resolution to each of these is the same as suggested in the Hybrid Approach section.
- BLU works better when given more CPU and memory than traditional row databases while co-existing on the same MPP database (and system) configuration as a traditional row schema and workload
- Depending on the performance objectives for the independent BLU workload, consideration may need to be given to some configuration changes to maximize performance.
- The storage enhancements in BLU may even allow a non-MPP configuration to be considered for this new workload.

DB2 BLU MPP support © Copyright IBM Corporation 2017

Implement BLU MPP, scenario 2: Existing DB2 MPP users with row-organized tables (independent)

The independent approach to adding column-organized tables into an existing DB2 MPP database, along with the applications using row-organized tables, is similar to the hybrid approach previously discussed. Some database configuration adjustments may be necessary. For example, BLU table support may require adjustments to some memory heaps, including utility heap memory and sort heap memory.

In some cases, the high degree of compression achieved with BLU tables may allow a workload to perform well on a non-MPP system.



IBM Training

Creating tables in a DB2 MPP database (1 of 2)

- In a DB2 MPP partitioned database tables are partitioned across the different servers based on a **distribution key**
  - A hash on the key fields determines the location of a row
- It is recommended that you specify a distribution key if you have enough information to choose an effective one
- If you do not specify a distribution key, DB2 will automatically select one for you
  - If your table has a primary key this will be used as the distribution key
  - Otherwise up to three columns will be selected automatically as a key

DB2 BLU MPP support © Copyright IBM Corporation 2017

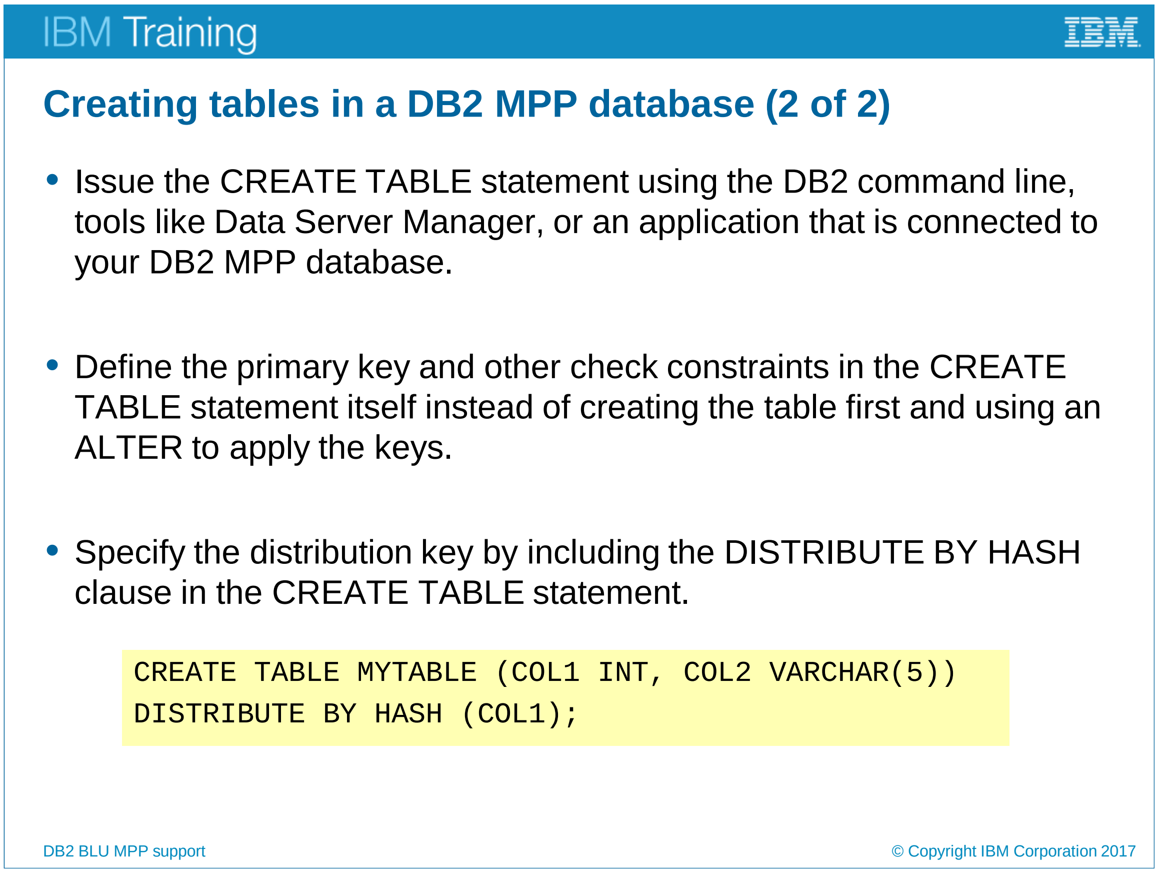
Creating tables in a DB2 MPP database

If you are moving from a non-MPP DB2 environment into a MPP database for BLU tables, there are some new factors involved in creating the tables.

In a DB2 MPP database, you need to decide how data will be distributed across partitions/servers in the MPP cluster.

Data is distributed via a "sharding" approach by hashing on a user-defined distribution key.

It is recommended that you pick your own distribution key for optimal performance. DB2 can also select one automatically for you if you do not specify one.



IBM Training

Creating tables in a DB2 MPP database (2 of 2)

- Issue the CREATE TABLE statement using the DB2 command line, tools like Data Server Manager, or an application that is connected to your DB2 MPP database.
- Define the primary key and other check constraints in the CREATE TABLE statement itself instead of creating the table first and using an ALTER to apply the keys.
- Specify the distribution key by including the DISTRIBUTE BY HASH clause in the CREATE TABLE statement.

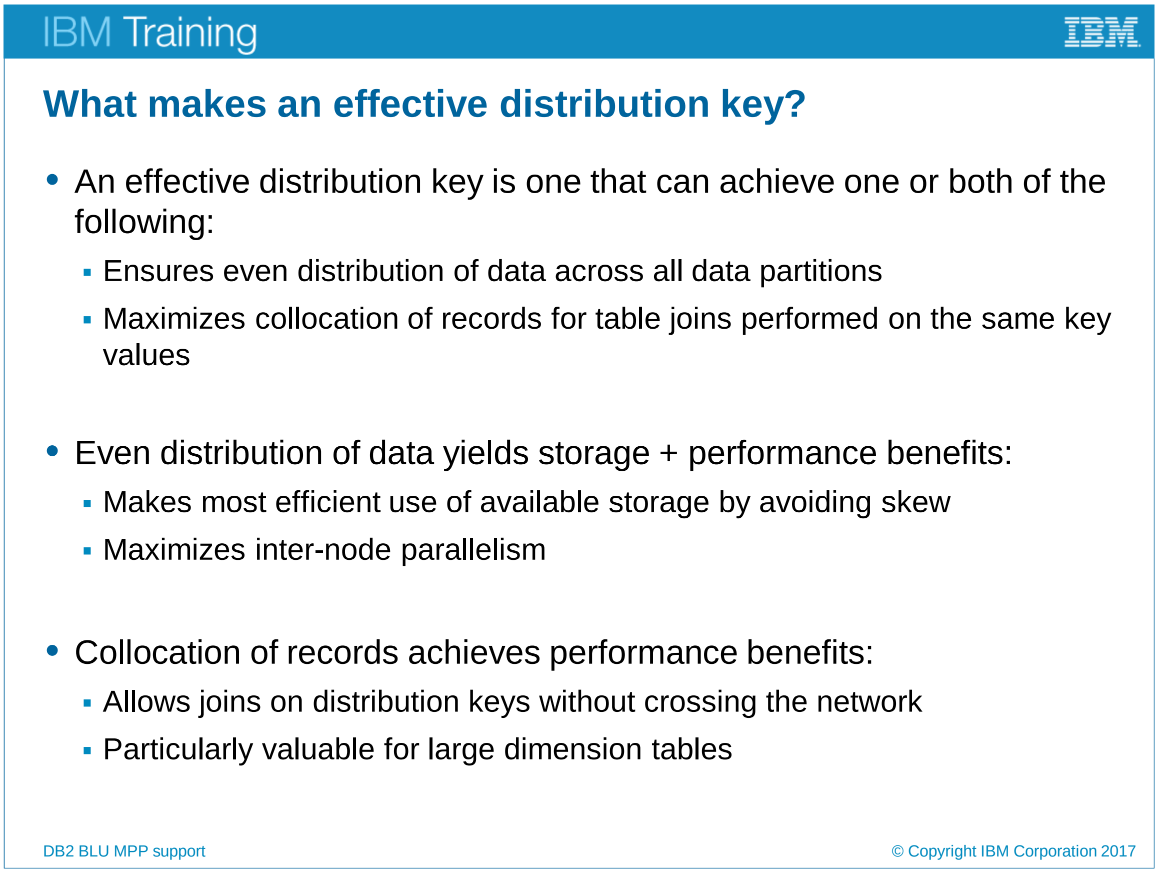
```
CREATE TABLE MYTABLE (COL1 INT, COL2 VARCHAR(5))
DISTRIBUTE BY HASH (COL1);
```

DB2 BLU MPP support

The steps in the slide outline how to create tables, including BLU tables, using a DB2 MPP database.

Note the recommendation to define keys on the CREATE TABLE statement itself. With a DB2 partitioned database, if you want to define a primary key or unique constraint for a table, the distribution key must either be the same as the unique key or a subset of the columns used for the unique or primary key.

The example in the slide shows how to specify a column (COL1) as the distribution key manually.



IBM Training

What makes an effective distribution key?

- An effective distribution key is one that can achieve one or both of the following:
  - Ensures even distribution of data across all data partitions
  - Maximizes collocation of records for table joins performed on the same key values
- Even distribution of data yields storage + performance benefits:
  - Makes most efficient use of available storage by avoiding skew
  - Maximizes inter-node parallelism
- Collocation of records achieves performance benefits:
  - Allows joins on distribution keys without crossing the network
  - Particularly valuable for large dimension tables

DB2 BLU MPP support

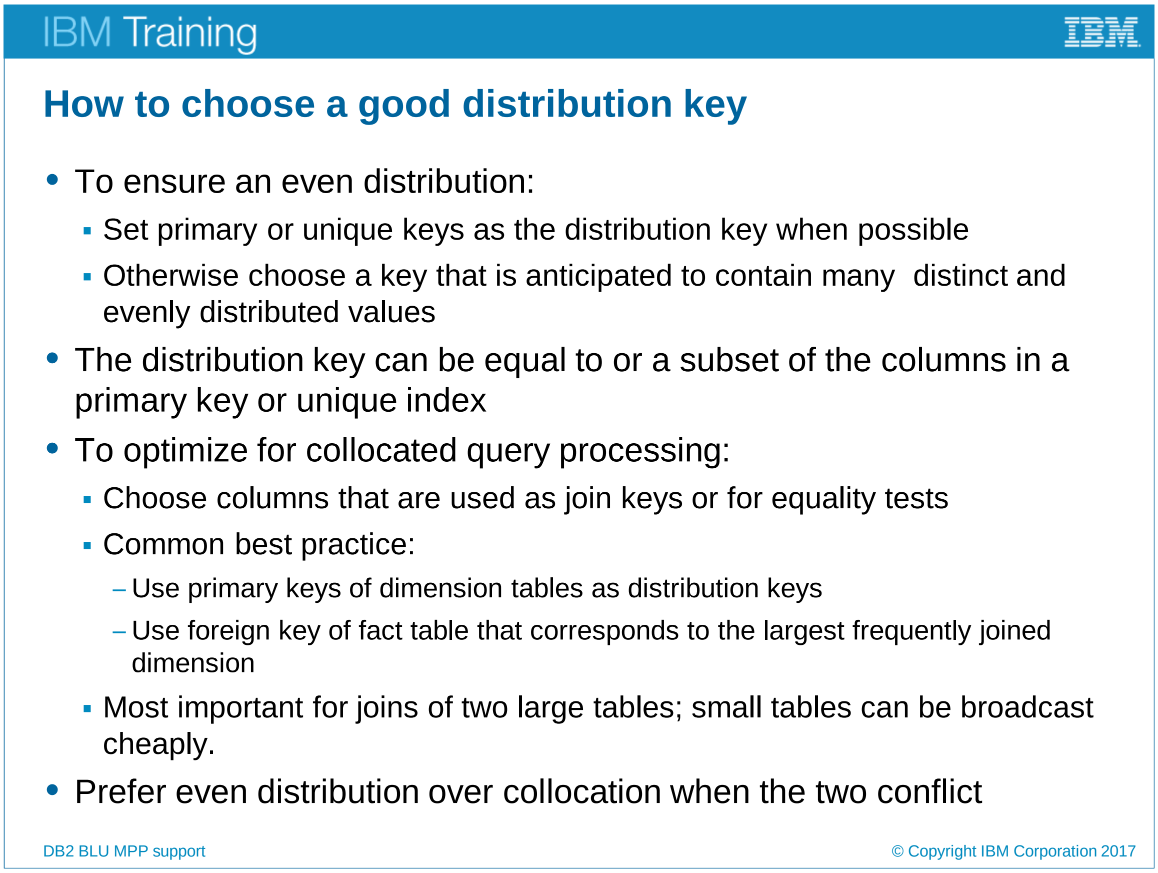
© Copyright IBM Corporation 2017

What makes an effective distribution key?

With a DB2 MPP database, selecting effective distribution key columns for tables can impact application performance.

Two of the primary goals in selecting distribution keys are:

- achieving an even distribution of data across all database partitions.
- improving table join efficiency. When tables are joined using columns that match the distribution keys for the tables, the join processing can utilize collocation. Collocated joins can join tables locally on each database partition without sending data to other partitions. This is particularly important for large dimension tables.


IBM Training

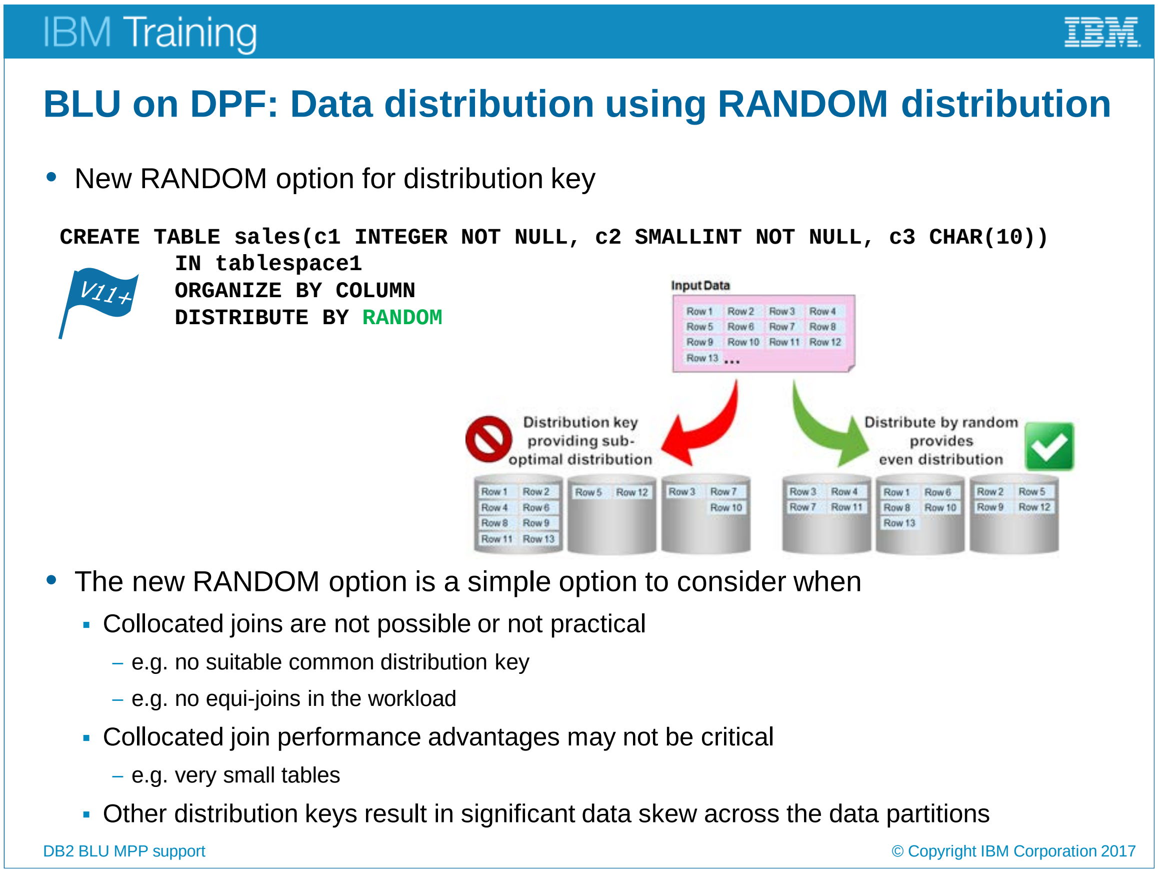
How to choose a good distribution key

- To ensure an even distribution:
  - Set primary or unique keys as the distribution key when possible
  - Otherwise choose a key that is anticipated to contain many distinct and evenly distributed values
- The distribution key can be equal to or a subset of the columns in a primary key or unique index
- To optimize for collocated query processing:
  - Choose columns that are used as join keys or for equality tests
  - Common best practice:
    - Use primary keys of dimension tables as distribution keys
    - Use foreign key of fact table that corresponds to the largest frequently joined dimension
  - Most important for joins of two large tables; small tables can be broadcast cheaply.
- Prefer even distribution over collocation when the two conflict

DB2 BLU MPP support © Copyright IBM Corporation 2017

How to choose a good distribution key

The slide summarizes common best practices for selecting a good distribution key for tables in a DB2 MPP database.



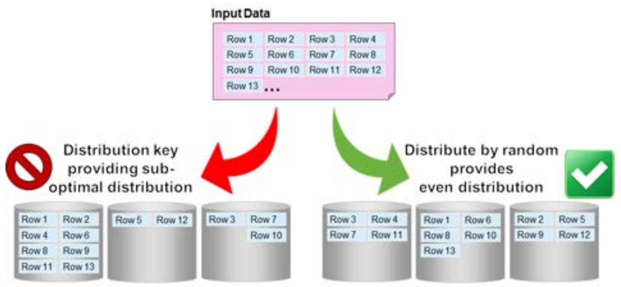
IBM Training

BLU on DPF: Data distribution using RANDOM distribution

- New RANDOM option for distribution key

```
CREATE TABLE sales(c1 INTEGER NOT NULL, c2 SMALLINT NOT NULL, c3 CHAR(10))
         IN tablespace1
         ORGANIZE BY COLUMN
         DISTRIBUTE BY RANDOM
```

V11+


Input Data
Row 1 Row 2 Row 3 Row 4
Row 5 Row 6 Row 7 Row 8
Row 9 Row 10 Row 11 Row 12
Row 13 ...
Distribution key providing sub-optimal distribution
Row 1 Row 2 Row 4 Row 6 Row 8 Row 9 Row 11 Row 13
Row 5 Row 12
Row 3 Row 7 Row 10
Distribute by random provides even distribution
Row 3 Row 4 Row 7 Row 11
Row 1 Row 6 Row 8 Row 10 Row 13
Row 2 Row 5 Row 9 Row 12

- The new RANDOM option is a simple option to consider when
  - Collocated joins are not possible or not practical
    - e.g. no suitable common distribution key
    - e.g. no equi-joins in the workload
  - Collocated join performance advantages may not be critical
    - e.g. very small tables
  - Other distribution keys result in significant data skew across the data partitions

DB2 BLU MPP support © Copyright IBM Corporation 2017

BLU on DPF: Data distribution using RANDOM distribution

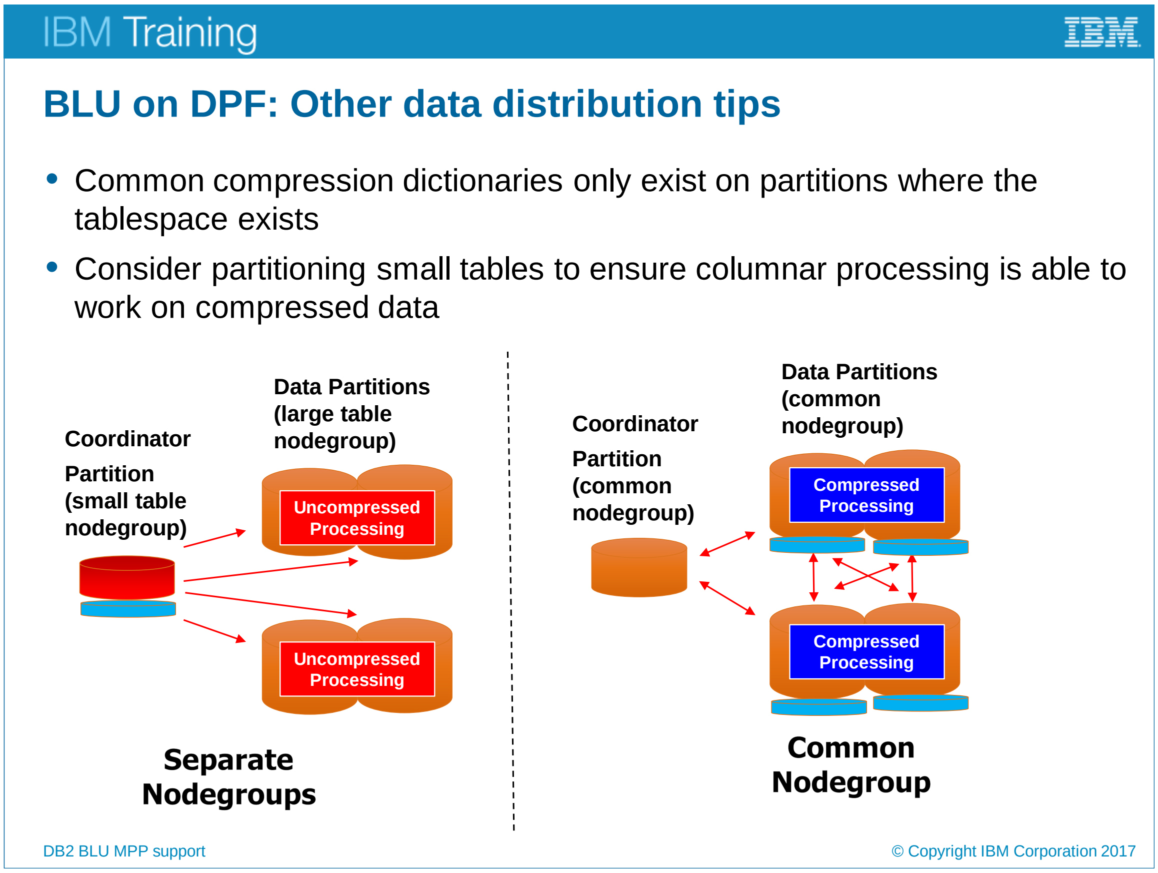
DB2 11.1 provides a new DISTRIBUTE BY RANDOM option for handling the distribution of data across multiple database partitions. This option could be used for BLU tables that match the conditions described on the slide.

In general this option would be used where even data distribution across multiple partitions is more important than collocation for multiple table joins.

If DISTRIBUTE BY RANDOM is specified, the distribution key is selected by the database manager in an effort to spread data evenly across all database partitions the table is defined on. There are two methods that the database manager uses to achieve this:

- **Random by unique:** If the table includes a unique or primary key, it uses the unique characteristics of the key columns to create a random spread of the data. The columns of the unique or primary key are used as the distribution keys.

- **Random by generation**: If the table does not have a unique or primary key, the database manager will include a column in the table to generate and store a generated value to use in the hashing function. The column will be created with the IMPLICITLY HIDDEN clause so that it does not appear in queries unless explicitly included. The value of the column will be automatically generated as new rows are added to the table. By default, the column name is RANDOM_DISTRIBUTION_KEY. If it collides with the existing column, a non-conflicting name will be generated by the database manager.



IBM Training

BLU on DPF: Other data distribution tips

- Common compression dictionaries only exist on partitions where the tablespace exists
- Consider partitioning small tables to ensure columnar processing is able to work on compressed data

DB2 BLU MPP support © Copyright IBM Corporation 2017

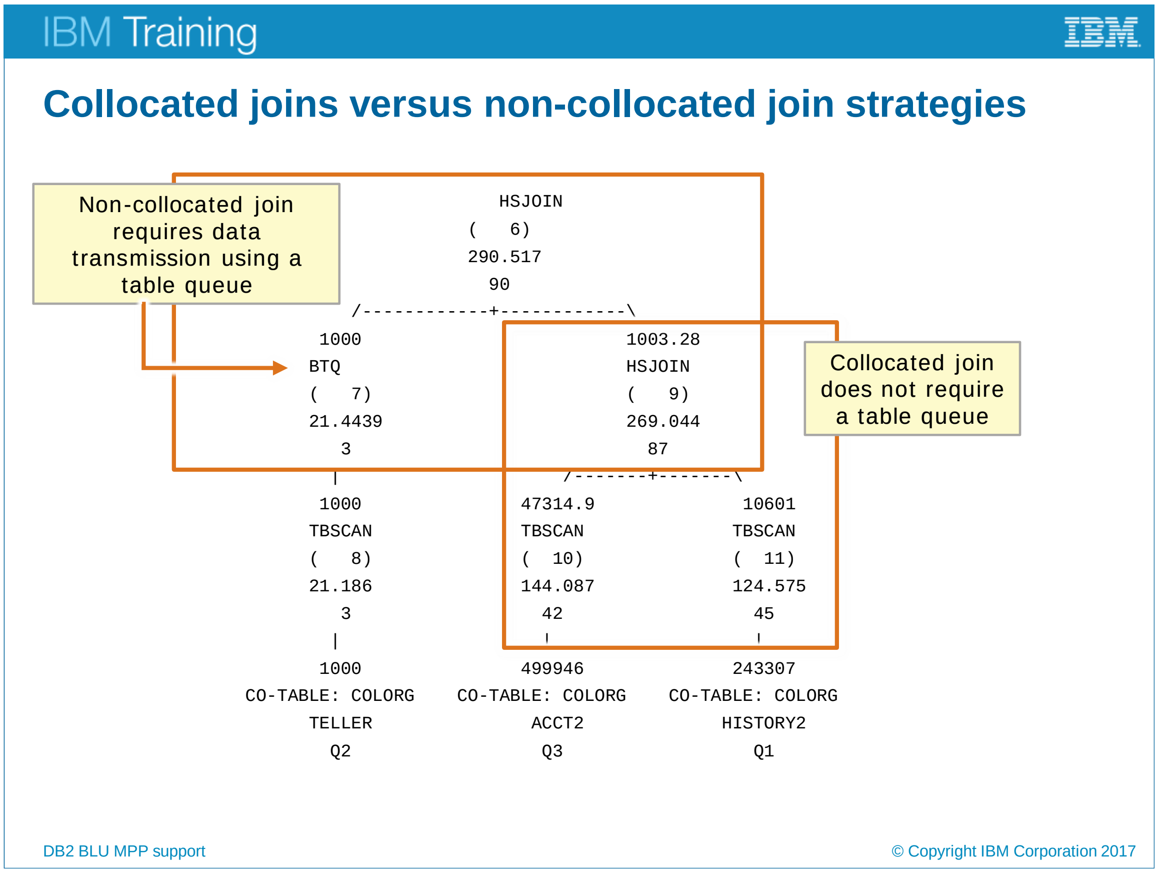
BLU on DPF: Other data distribution tips

A common practice with row-organized tables in DB2 MPP partitioned databases is to utilize a single database to store small tables.

With BLU MPP, the use of common column compression dictionary data may offer an advantage to having the small tables span multiple partitions.

Since a BLU table will only have the common dictionary data on the partition where the table is stored, if a small table is joined to a large multiple partition table, if the optimizer chooses to send the small table data using a table queue to multiple database partitions, having the small table's common dictionary on those partitions allows the small table to remain encoded as it is sent across partitions.

This slide shows how spreading a small table across the same partitions as the large tables that are commonly joined together allows the joins to operate on the compressed encoded data. Having the table defined in table spaces with a common database partition group, or node group, will ensure that the table dictionaries have common partition locations.



IBM Training

Collocated joins versus non-collocated join strategies

```
                              HSJOIN
                             (   6)
                             290.517
                               90
                  /------------+-----------\
             1000                           1003.28
             BTQ                            HSJOIN
             (   7)                         (   9)
             21.4439                        269.044
               3                              87
               |                       /-------+-------\
             1000                  47314.9            10601
             TBSCAN                TBSCAN             TBSCAN
             (   8)                (  10)             (  11)
             21.186                144.087            124.575
               3                     42                 45
               |                     |                  |
             1000                  499946             243307
      CO-TABLE: COLORG      CO-TABLE: COLORG      CO-TABLE: COLORG
             TELLER                ACCT2              HISTORY2
               Q2                   Q3                  Q1
```

Non-collocated join requires data transmission using a table queue
Collocated join does not require a table queue

DB2 BLU MPP support © Copyright IBM Corporation 2017

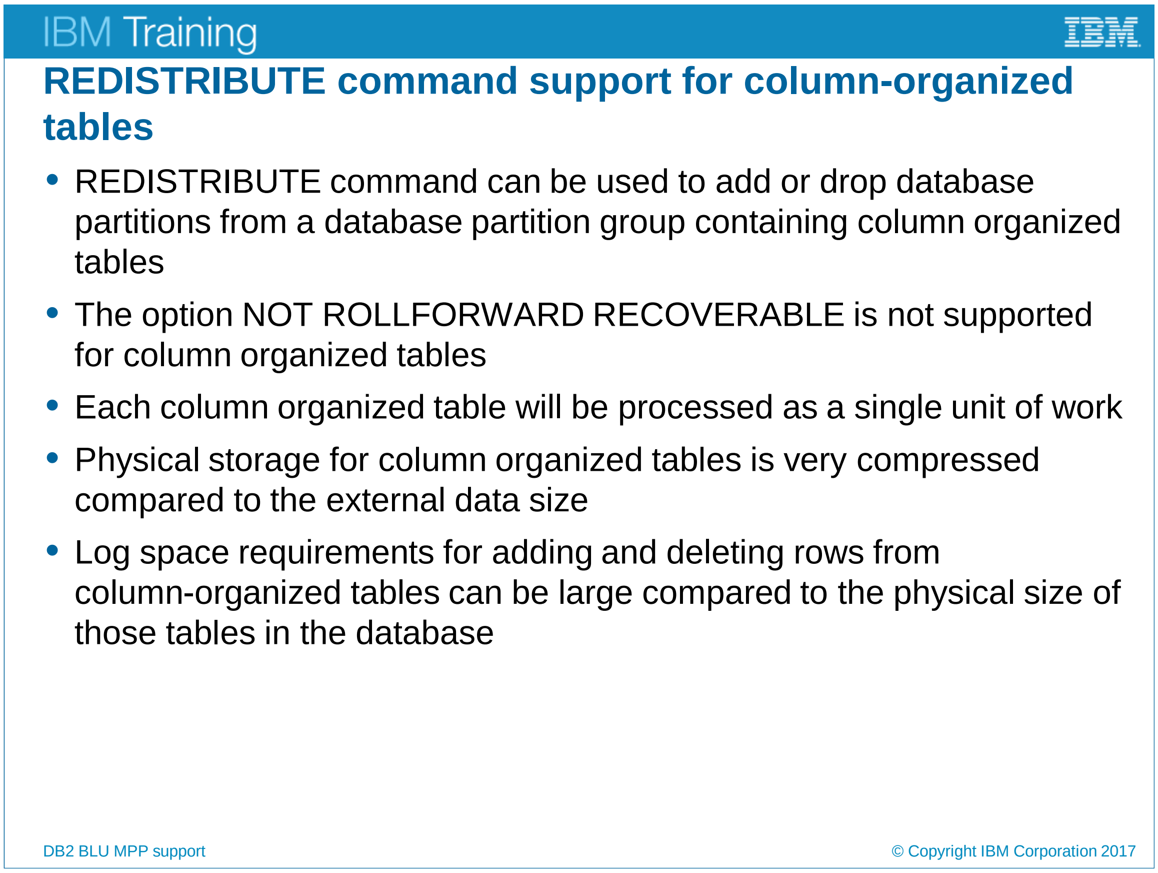
Collocated joins versus non-collocated join strategies

This slide shows a portion of the access plan report form a DB2 explain tool, showing the table join processing for joining three column-organized tables using a multiple partition DB2 database.

The two tables ACCT2 and HISTORY2 are joined using a collocated hash join. This is possible because of the following characteristics:

- The two tables were defined with a common database partition group.
- The two tables were created with matched distribution keys defined on the column or columns that are used as the join predicates for these tables in the SQL statement.

The joining of the composite result of the first join with the TELLER table is a non-collocated join. The BTQ, or Broadcast Table Queue, plan step 7, is used to move data retrieved from the TELLER table to the partitions where the other two tables were joined in order to perform the next hash join, access plan step 6.


IBM Training

REDISTRIBUTE command support for column-organized tables

- REDISTRIBUTE command can be used to add or drop database partitions from a database partition group containing column organized tables
- The option NOT ROLLFORWARD RECOVERABLE is not supported for column organized tables
- Each column organized table will be processed as a single unit of work
- Physical storage for column organized tables is very compressed compared to the external data size
- Log space requirements for adding and deleting rows from column-organized tables can be large compared to the physical size of those tables in the database

DB2 BLU MPP support © Copyright IBM Corporation 2017

REDISTRIBUTE command support for column organized tables

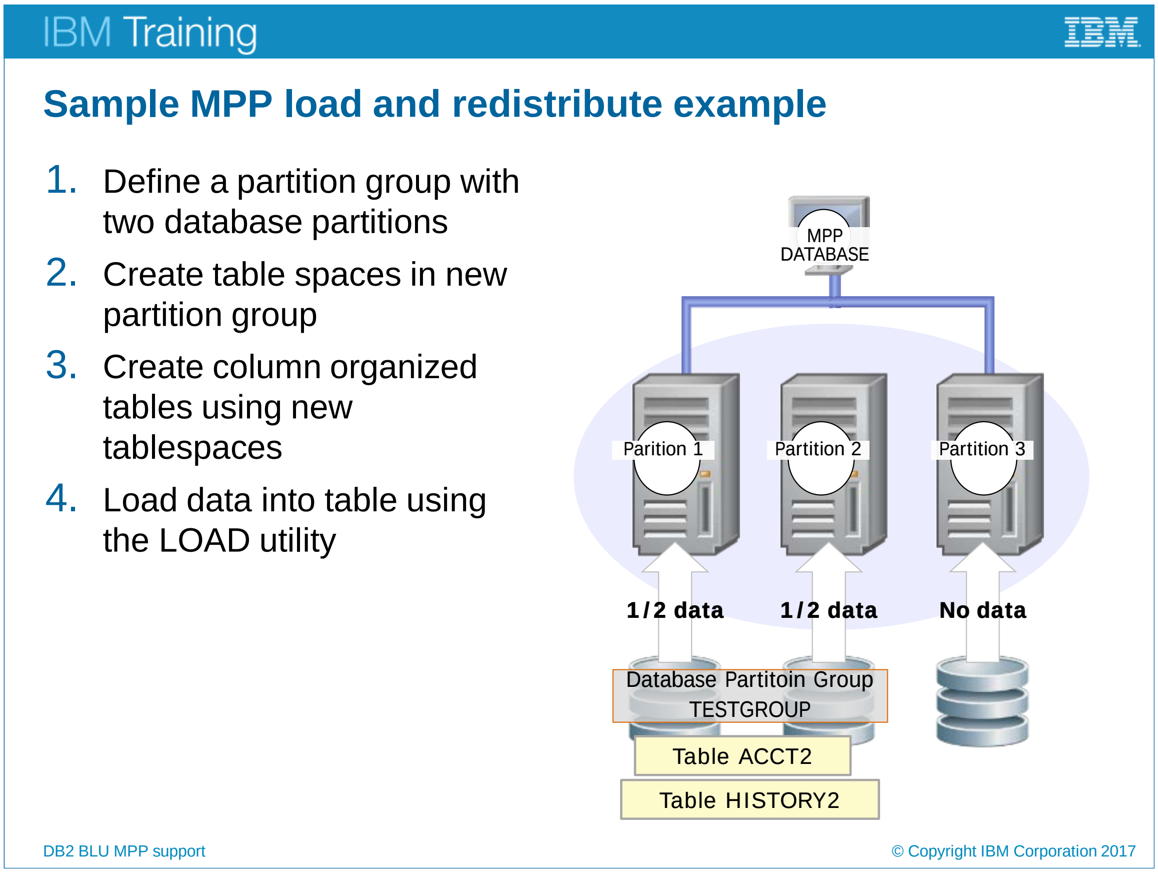
Data redistribution is a database administration operation that can be performed to primarily move data within a partitioned database environment when partitions are added or removed. The goal of this operation is typically to balance the usage of storage space, improve database system performance, or satisfy other system requirements. Data redistribution can be performed by using the command REDISTRIBUTE DATABASE PARTITION GROUP.

Data redistribution within a partitioned database is done for one of the following reasons:

- To rebalance data whenever a new database partition is added to the database environment or an existing database partition is removed.
- To introduce user-specific data distribution across partitions.
- To secure sensitive data by isolating it within a particular partition.

With DB2 11.1, the BLU tables in a DB2 MPP database can utilize the REDISTRIBUTE DATABASE PARTITION GROUP command, but the NOT ROLLFORWARD RECOVERABLE option is not currently supported.

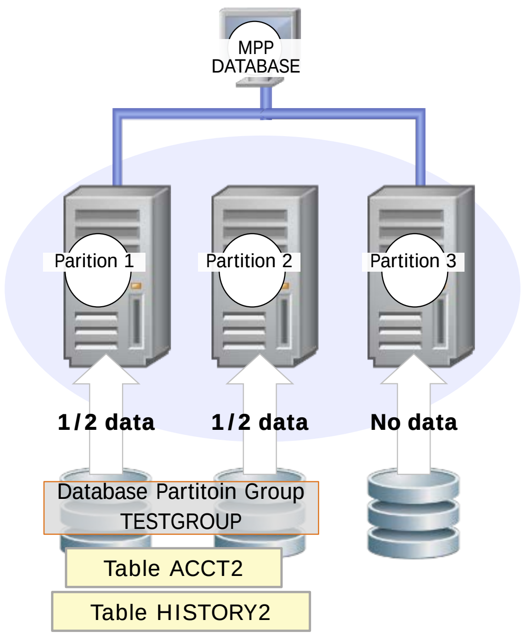
Each table that is moved by REDISTRIBUTE is considered one unit of work, which can be rolled back if there is a system failure or other problem. This means that the logging configuration for the database will need to support the insert and delete activity for the largest table being moved. Since BLU tables are stored in a very compressed format, the log requirements can exceed the physical storage required for the table.



IBM Training

Sample MPP load and redistribute example

1. Define a partition group with two database partitions
2. Create table spaces in new partition group
3. Create column organized tables using new tablespaces
4. Load data into table using the LOAD utility



DB2 BLU MPP support © Copyright IBM Corporation 2017

Sample MPP load and redistribute example

We are going to step through a series of steps, demonstrating the loading of several BLU tables in a DB2 MPP database and then using REDISTRIBUTE to spread the table data to an additional database partition.

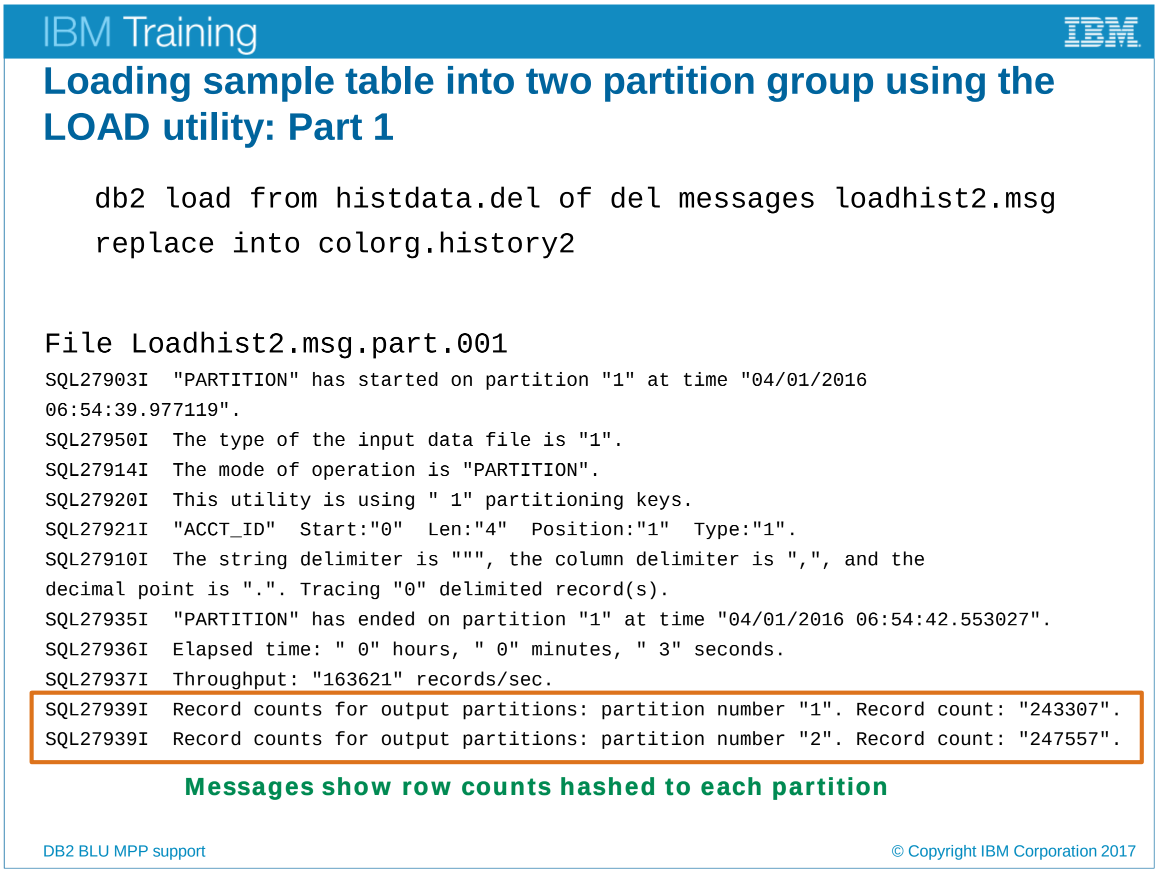
In a DB2 MPP database, a database object, the database partition group, defines a set of database partitions. We are going to start with a database partition group, named TESTGROUP, which maps to partitions 1 and 2.

If we create table spaces to hold the new BLU table data, we can assign those table spaces to the database partition group.

Any tables that are created in the new table spaces will have the data distributed between partitions 1 and 2.

The DB2 LOAD utility can take input data and partition the incoming data for parallel load processing across multiple partitions.

We will load two tables, ACCT2 and HISTORY2. We expect roughly 50 percent of each table to be stored on each of the two database partitions.



IBM Training

Loading sample table into two partition group using the LOAD utility: Part 1

```
db2 load from histdata.del of del messages loadhist2.msg
replace into colorg.history2
```

```
File Loadhist2.msg.part.001
SQL27903I  "PARTITION" has started on partition "1" at time "04/01/2016
06:54:39.977119".
SQL27950I  The type of the input data file is "1".
SQL27914I  The mode of operation is "PARTITION".
SQL27920I  This utility is using " 1" partitioning keys.
SQL27921I  "ACCT_ID"  Start:"0"  Len:"4"  Position:"1"  Type:"1".
SQL27910I  The string delimiter is """, the column delimiter is ",", and the
decimal point is ".". Tracing "0" delimited record(s).
SQL27935I  "PARTITION" has ended on partition "1" at time "04/01/2016 06:54:42.553027".
SQL27936I  Elapsed time: " 0" hours, " 0" minutes, " 3" seconds.
SQL27937I  Throughput: "163621" records/sec.
SQL27939I  Record counts for output partitions: partition number "1". Record count: "243307".
SQL27939I  Record counts for output partitions: partition number "2". Record count: "247557".
```

Messages show row counts hashed to each partition

DB2 BLU MPP support

© Copyright IBM Corporation 2017

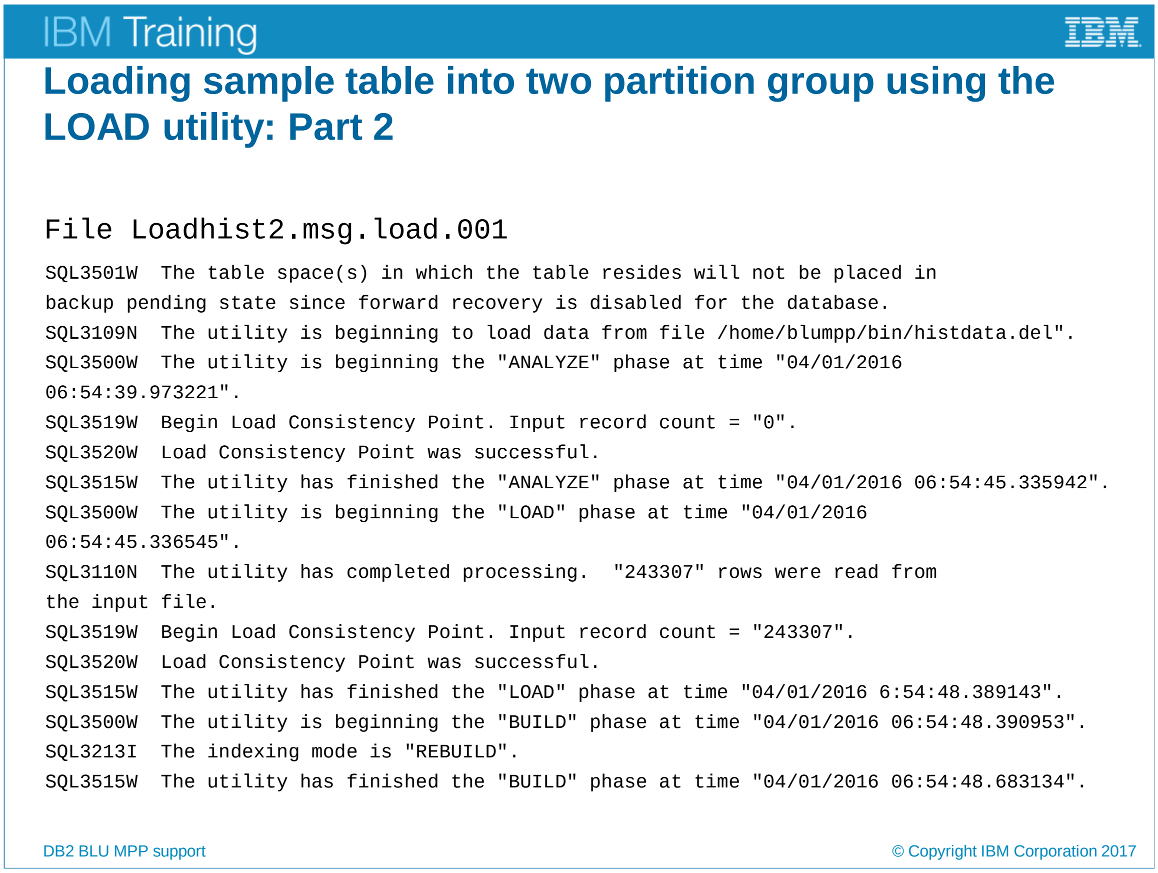
Loading sample table into two partition group using the LOAD utility: Part 1

When we execute the DB2 LOAD utility on a multiple partition DB2 database, a set of message files are generated.

The sample output shown in the slide is the partitioning report.

The report includes:

- The column(s) used to partition the data. In this case, one column (ACCT_ID) is used.
- The report shows a throughput for partition processing in rows per second.
- The record counts sent to each database partition. In this example, the record counts are pretty well balanced between to two partitions.


IBM Training

Loading sample table into two partition group using the LOAD utility: Part 2

File Loadhist2.msg.load.001

```
SQL3501W  The table space(s) in which the table resides will not be placed in
backup pending state since forward recovery is disabled for the database.
SQL3109N  The utility is beginning to load data from file /home/blumpp/bin/histdata.del".
SQL3500W  The utility is beginning the "ANALYZE" phase at time "04/01/2016
06:54:39.973221".
SQL3519W  Begin Load Consistency Point. Input record count = "0".
SQL3520W  Load Consistency Point was successful.
SQL3515W  The utility has finished the "ANALYZE" phase at time "04/01/2016 06:54:45.335942".
SQL3500W  The utility is beginning the "LOAD" phase at time "04/01/2016
06:54:45.336545".
SQL3110N  The utility has completed processing.  "243307" rows were read from
the input file.
SQL3519W  Begin Load Consistency Point. Input record count = "243307".
SQL3520W  Load Consistency Point was successful.
SQL3515W  The utility has finished the "LOAD" phase at time "04/01/2016 6:54:48.389143".
SQL3500W  The utility is beginning the "BUILD" phase at time "04/01/2016 06:54:48.390953".
SQL3213I  The indexing mode is "REBUILD".
SQL3515W  The utility has finished the "BUILD" phase at time "04/01/2016 06:54:48.683134".
```

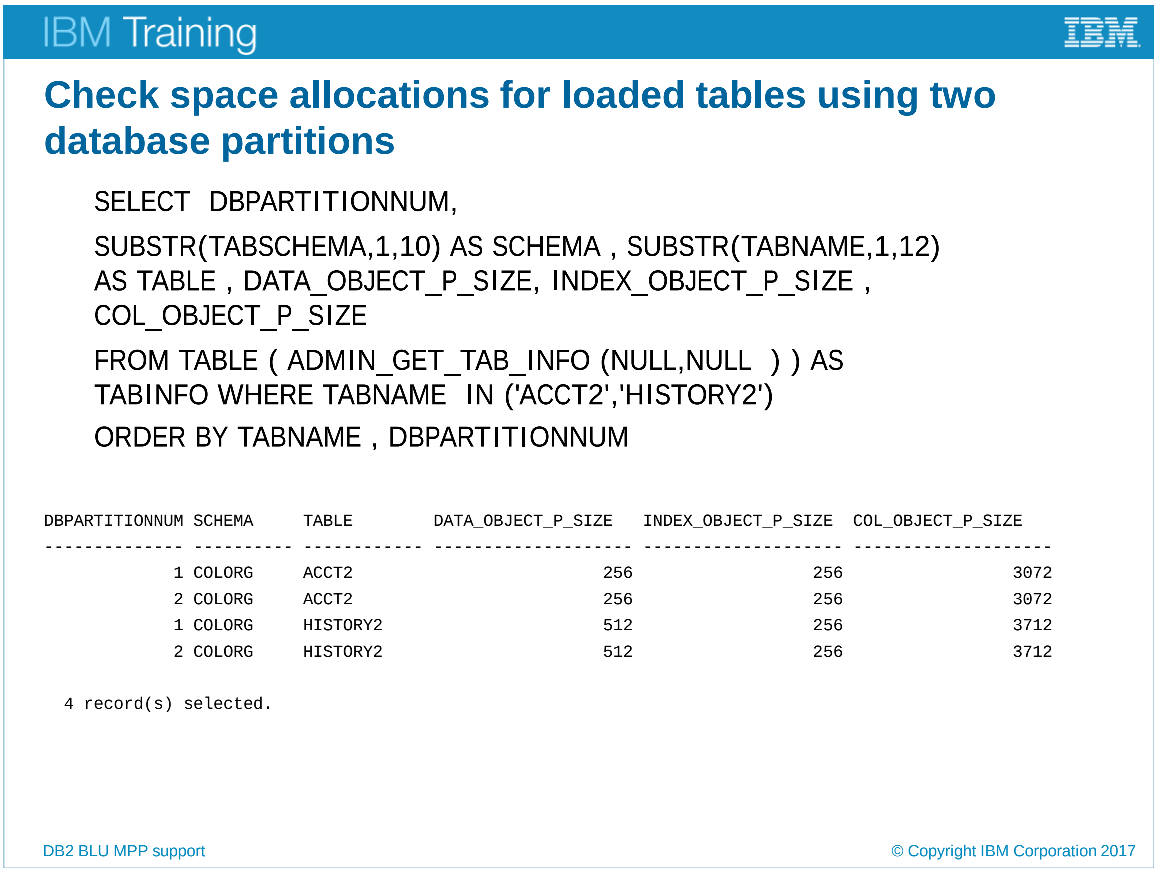
DB2 BLU MPP support © Copyright IBM Corporation 2017

Loading sample table into two partition group using the LOAD utility: Part 2

The DB2 LOAD utility also produces one report per database partition to describe the loading of data onto that partition.

Since we are loading BLU tables that are empty, LOAD will include the ANALYZE phase, used to create column dictionary data.

The LOAD messages also show processing for the LOAD and BUILD phases of processing. The page map index data for each database partition is local to the table data on a partition.



IBM Training

Check space allocations for loaded tables using two database partitions

```
SELECT  DBPARTITIONNUM,
SUBSTR(TABSCHEMA,1,10) AS SCHEMA , SUBSTR(TABNAME,1,12)
AS TABLE , DATA_OBJECT_P_SIZE, INDEX_OBJECT_P_SIZE ,
COL_OBJECT_P_SIZE
FROM TABLE ( ADMIN_GET_TAB_INFO (NULL,NULL  ) ) AS
TABINFO WHERE TABNAME  IN ('ACCT2','HISTORY2')
ORDER BY TABNAME , DBPARTITIONNUM
```

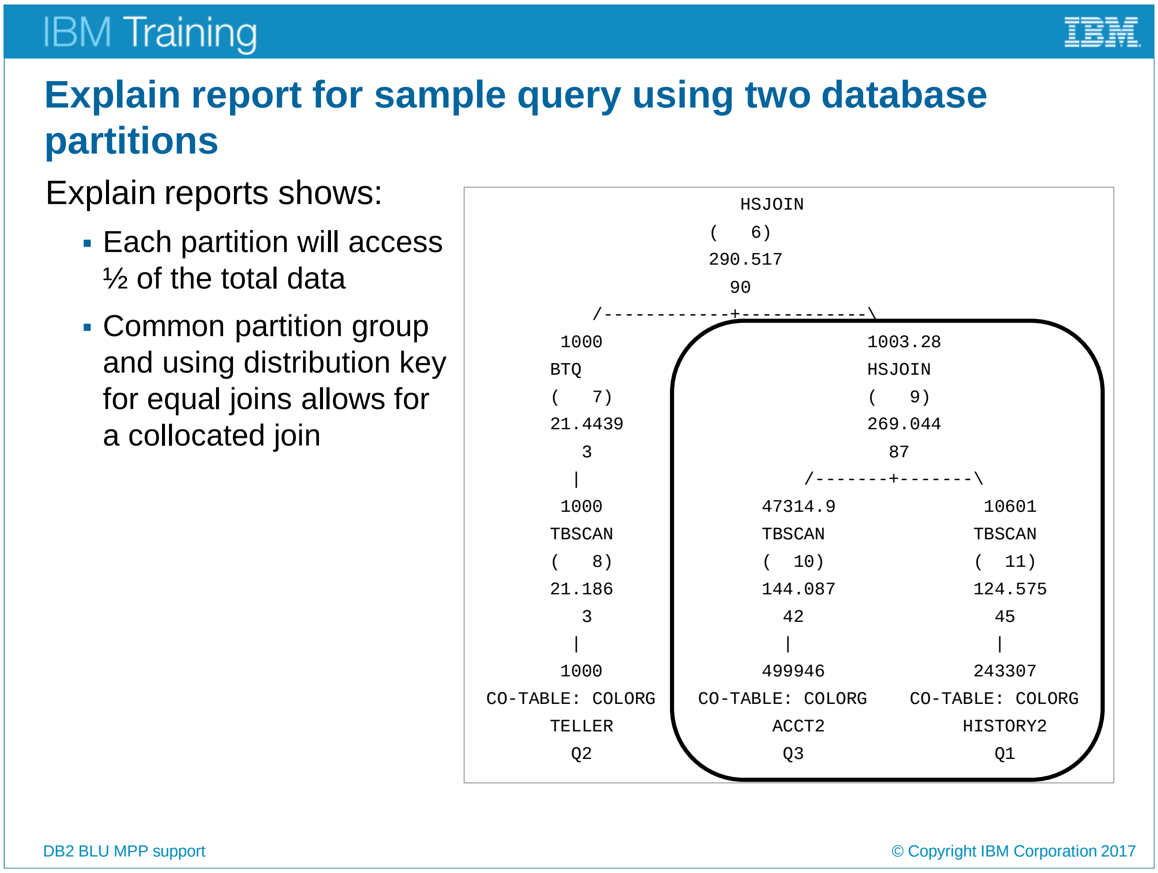
```
DBPARTITIONNUM SCHEMA     TABLE        DATA_OBJECT_P_SIZE   INDEX_OBJECT_P_SIZE  COL_OBJECT_P_SIZE
-------------- ---------- ------------ -------------------- -------------------- --------------------
             1 COLORG     ACCT2                         256                  256                 3072
             2 COLORG     ACCT2                         256                  256                 3072
             1 COLORG     HISTORY2                      512                  256                 3712
             2 COLORG     HISTORY2                      512                  256                 3712

  4 record(s) selected.
```

DB2 BLU MPP support © Copyright IBM Corporation 2017

Check space allocations for loaded tables using two database partitions

Now that we have loaded two column-organized tables, we can use DB2 monitoring facilities like the table function ADMIN_GET_TAB_INFO to query the physical storage allocations associated with the data, index, and column-organized objects for each table on each database partition.

With DB2 BLU tables, we expect the largest allocation of disk space to be associated with the Column Object.



IBM Training

Explain report for sample query using two database partitions

Explain reports shows:

- Each partition will access ½ of the total data
- Common partition group and using distribution key for equal joins allows for a collocated join

```
                         HSJOIN
                      (    6)
                      290.517
                        90
              /------------+------------\
           1000                        1003.28
          BTQ                          HSJOIN
          (    7)                      (    9)
          21.4439                      269.044
            3                            87
            |                   /-------+-------\
           1000              47314.9            10601
          TBSCAN             TBSCAN             TBSCAN
          (    8)            (   10)            (   11)
          21.186             144.087            124.575
            3                  42                 45
            |                  |                  |
           1000              499946             243307
   CO-TABLE: COLORG    CO-TABLE: COLORG    CO-TABLE: COLORG
          TELLER              ACCT2              HISTORY2
           Q2                  Q3                  Q1
```

DB2 BLU MPP support

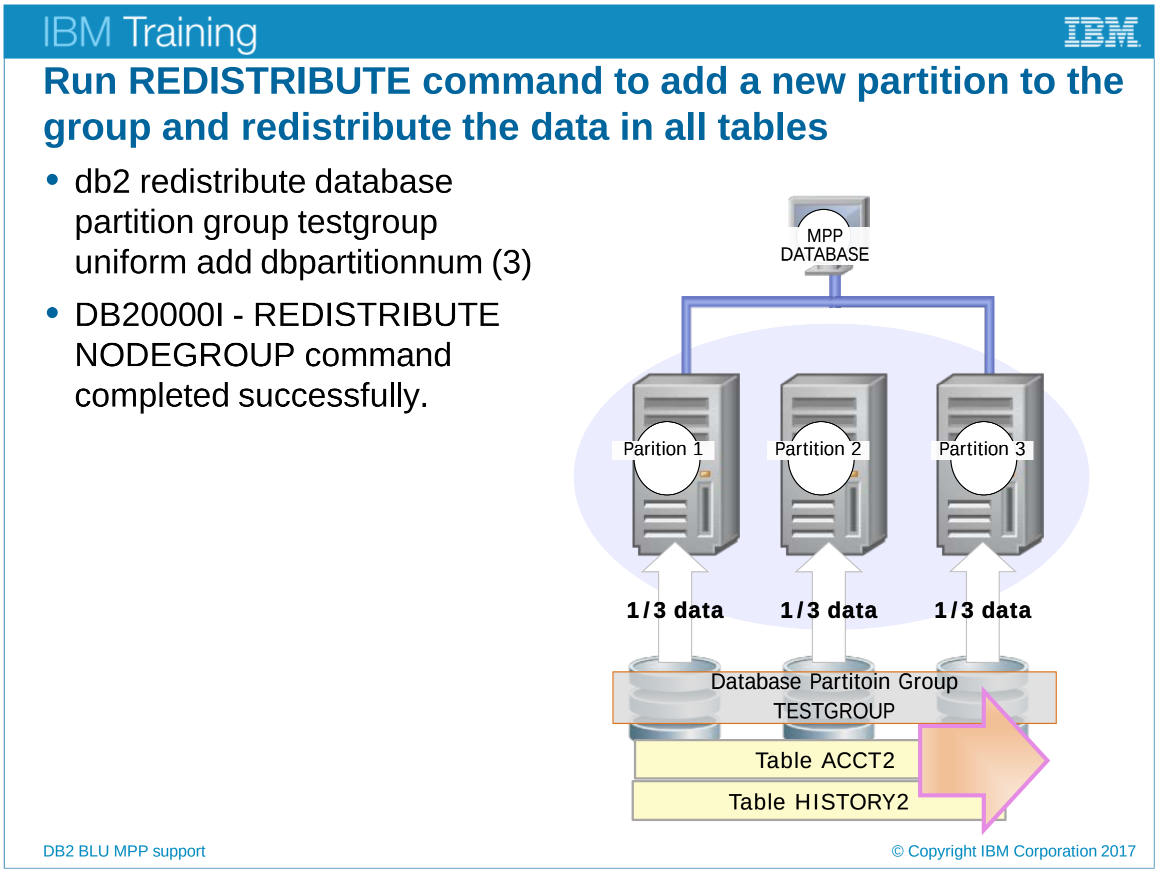
© Copyright IBM Corporation 2017

Explain report for sample query using two database partitions

The DB2 Optimizer uses the database statistics from one database partition to estimate the query processing at each database partition, assuming that parallel processing for each partition will be the same.

In the sample portion of the explain report, the cardinality estimates, shown above each operation, indicate that DB2 expects each of the two partitions to process 50 percent of the total data, and execute in parallel (for example, we know from an earlier chart that the partitioning phase for the HISTORY2 table sent 243307 rows to just partition 1. Clearly, partition 1 holds the statistics for this table per the Explain output. The optimizer assumes each partition to have the same row count, which is a good approximation).

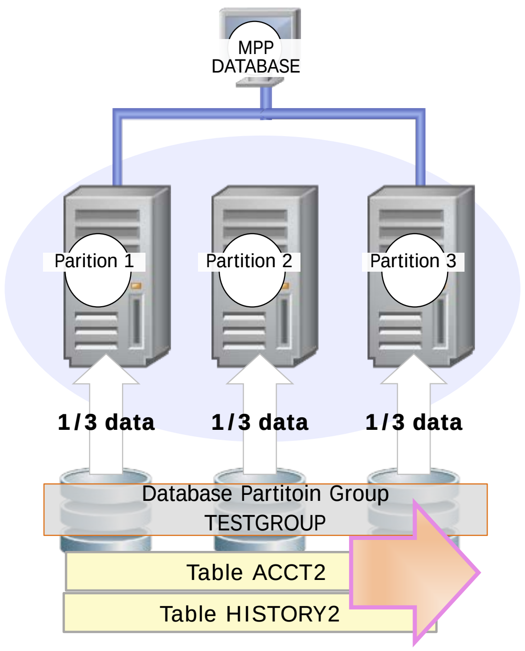
A third table is also referenced in this sample query. Since the two tables ACCT2 and HISTORY2 are in the same database partition group and they are being joined using the distribution key column, ACCT_ID, a collocated join is performed.



IBM Training

Run REDISTRIBUTE command to add a new partition to the group and redistribute the data in all tables

- db2 redistribute database partition group testgroup uniform add dbpartitionnum (3)
- DB20000I - REDISTRIBUTE NODEGROUP command completed successfully.



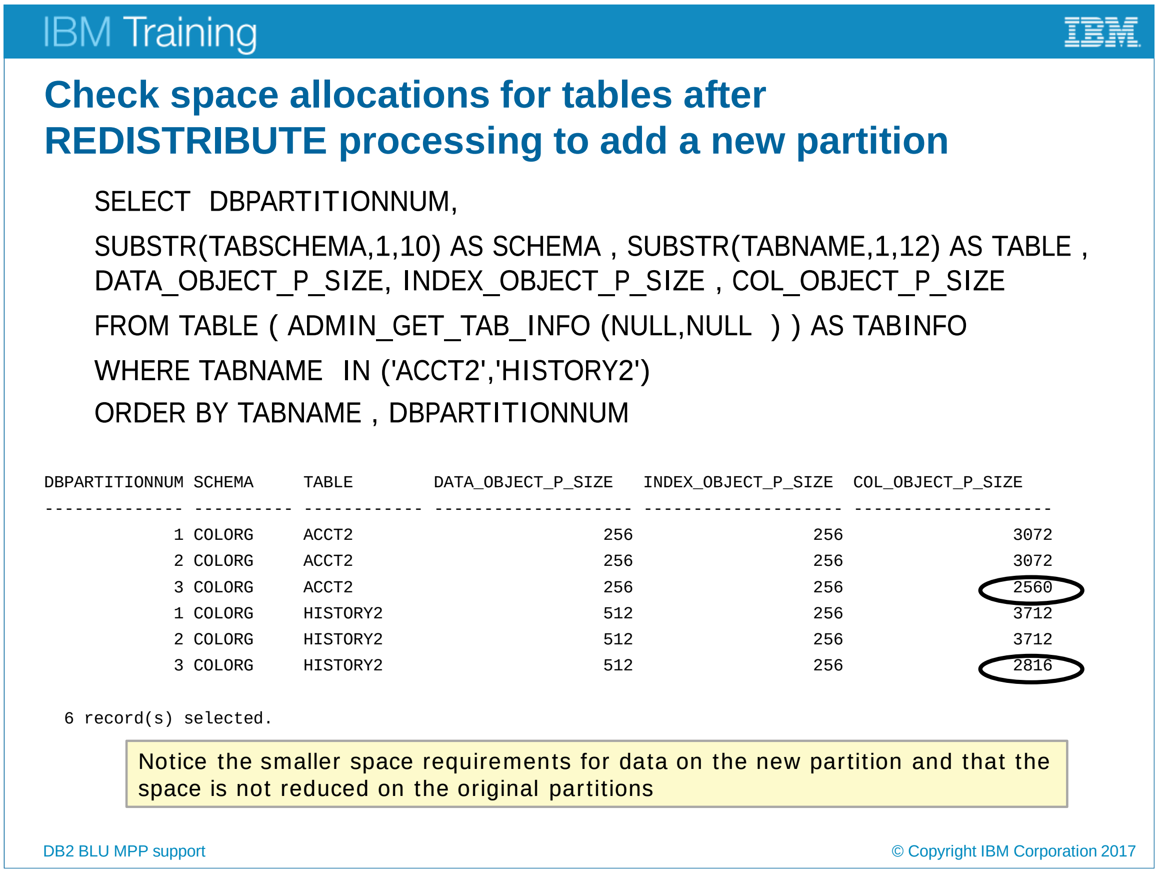
DB2 BLU MPP support © Copyright IBM Corporation 2017

Run REDISTRIBUTE command to add a new partition to the group and redistribute the data in all tables

If a new database partition was added to the DB2 MPP cluster, we could extend the database partition group we created, TESTGROUP, to include the new partition.

The example REDISTRIBUTE command shown on the slide adds a new partition (number 3) to the database partition group and distributes the data in the two tables we created (ACCT2 and HISTORY2) to be uniformly balanced.

In this case, some rows from the two tables will be deleted from partitions 1 and 2 and inserted into storage on the new partition 3.



IBM Training

Check space allocations for tables after REDISTRIBUTE processing to add a new partition

```
SELECT  DBPARTITIONNUM,
SUBSTR(TABSCHEMA,1,10) AS SCHEMA , SUBSTR(TABNAME,1,12) AS TABLE ,
DATA_OBJECT_P_SIZE, INDEX_OBJECT_P_SIZE , COL_OBJECT_P_SIZE
FROM TABLE ( ADMIN_GET_TAB_INFO (NULL,NULL  ) ) AS TABINFO
WHERE TABNAME  IN ('ACCT2','HISTORY2')
ORDER BY TABNAME , DBPARTITIONNUM
```

```
DBPARTITIONNUM SCHEMA     TABLE        DATA_OBJECT_P_SIZE   INDEX_OBJECT_P_SIZE  COL_OBJECT_P_SIZE
-------------- ---------- ------------ -------------------- -------------------- --------------------
             1 COLORG     ACCT2                         256                  256                 3072
             2 COLORG     ACCT2                         256                  256                 3072
             3 COLORG     ACCT2                         256                  256                 2560
             1 COLORG     HISTORY2                      512                  256                 3712
             2 COLORG     HISTORY2                      512                  256                 3712
             3 COLORG     HISTORY2                      512                  256                 2816

  6 record(s) selected.
```

Notice the smaller space requirements for data on the new partition and that the space is not reduced on the original partitions

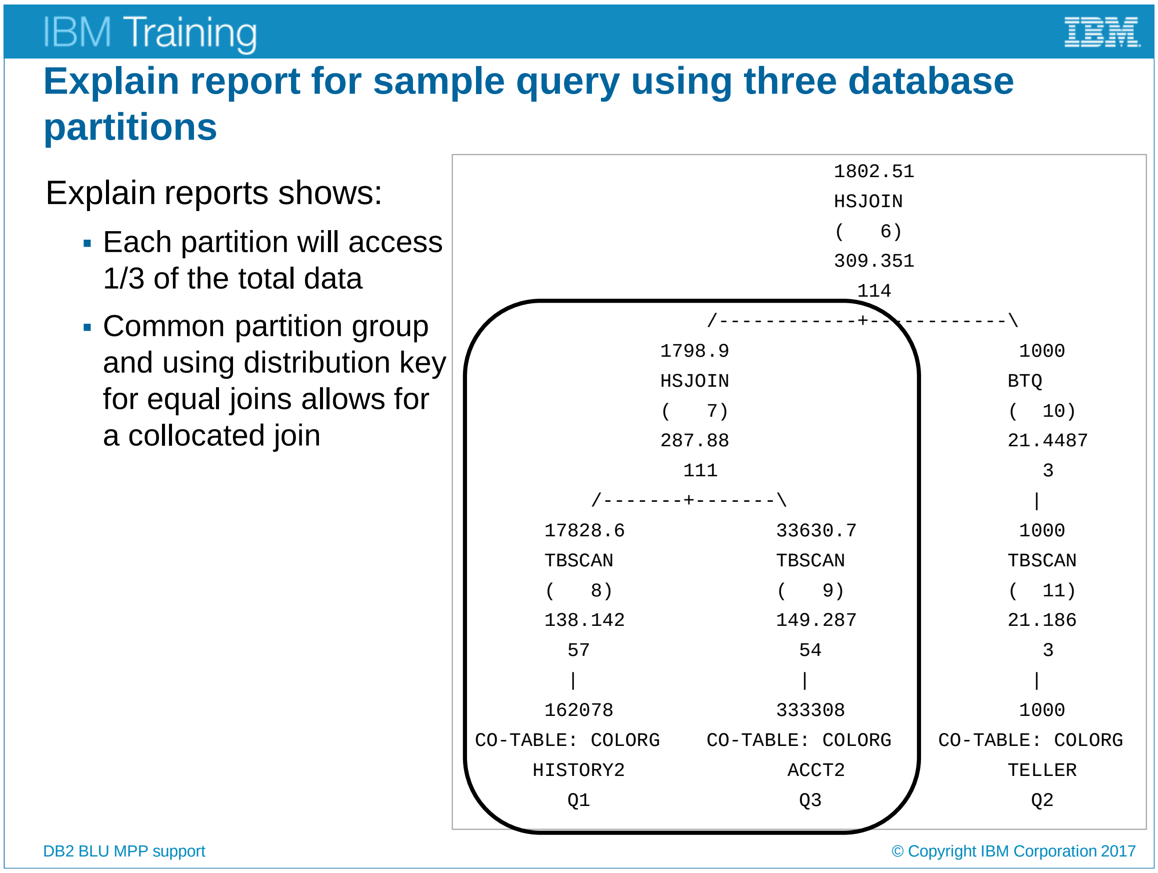
DB2 BLU MPP support © Copyright IBM Corporation 2017

Check space allocations for tables after REDISTRIBUTE processing to add a new partition

When the REDISTRIBUTE command (shown on the previous slide) completes, we can query the disk space allocations for the two tables, which are now stored on three database partitions.

The sample query result shows that the space allocation on the new database partition is smaller because it was given roughly one third of the data for each table.

The space allocation on the two original partitions did not change as a result of REDISTRIBUTE processing. Deleting rows of data in a column-organized table marks the data as deleted but does not release any space for reuse.



IBM Training

Explain report for sample query using three database partitions

Explain reports shows:

- Each partition will access 1/3 of the total data
- Common partition group and using distribution key for equal joins allows for a collocated join

```
                              1802.51
                              HSJOIN
                              (   6)
                              309.351
                                114
                 /------------+-------------\
           1798.9                             1000
           HSJOIN                             BTQ
           (   7)                             (  10)
           287.88                             21.4487
            111                                 3
      /-------+-------\                         |
17828.6             33630.7                   1000
TBSCAN              TBSCAN                    TBSCAN
(   8)              (   9)                    (  11)
138.142             149.287                   21.186
  57                  54                        3
  |                   |                         |
162078              333308                    1000
CO-TABLE: COLORG    CO-TABLE: COLORG    CO-TABLE: COLORG
  HISTORY2            ACCT2               TELLER
    Q1                  Q3                  Q2
```

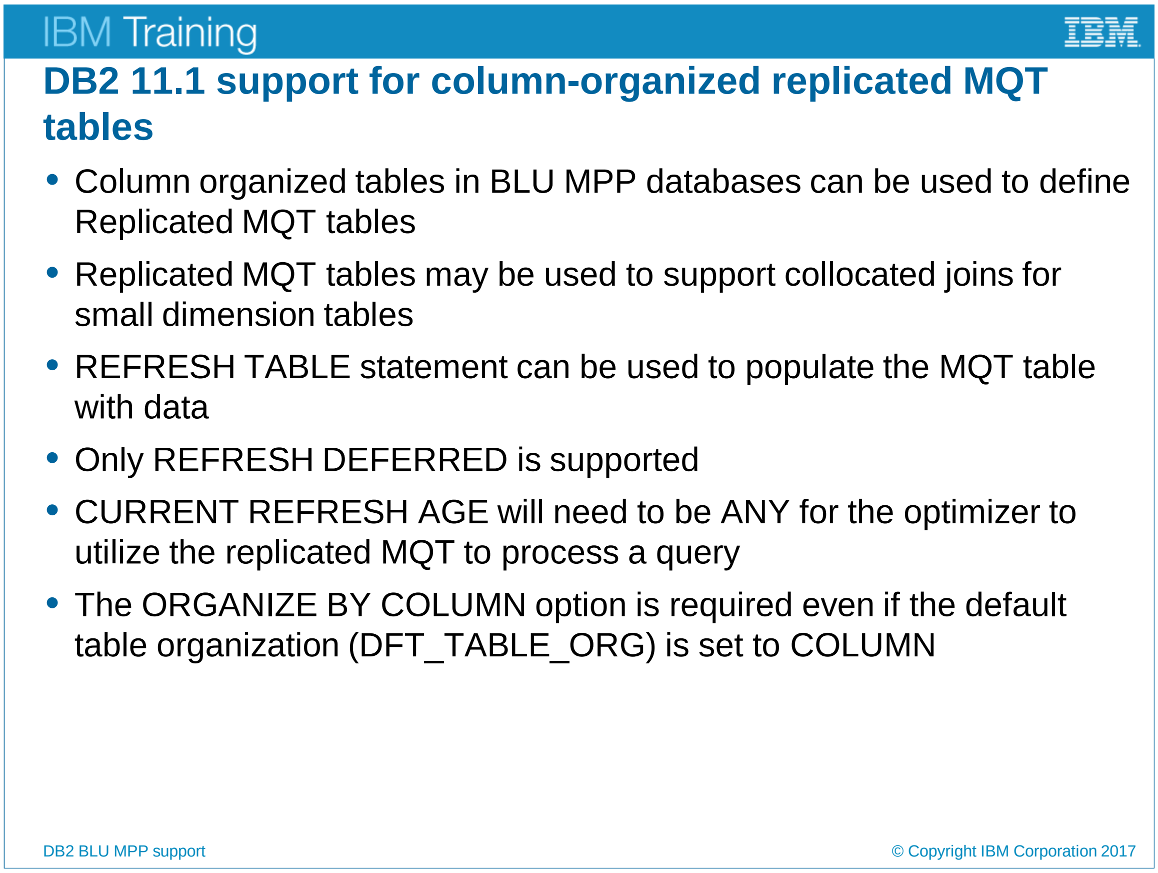
DB2 BLU MPP support © Copyright IBM Corporation 2017

Explain report for sample query using three database partitions

This slide shows an updated access plan diagram produced by a DB2 explain tool for the same SQL query used for the previous example.

After the two tables were redistributed to utilize three database partitions rather than the two original partitions, the optimizer finds a smaller amount of data associated with each partition.

Now the report shows that roughly 330,000 of the one million rows in the ACCT2 table will be scanned by each database partition. The performance of the SQL query should now be improved as the resources of three database servers can now process the query in parallel to produce the result.



IBM Training

DB2 11.1 support for column-organized replicated MQT tables

- Column organized tables in BLU MPP databases can be used to define Replicated MQT tables
- Replicated MQT tables may be used to support collocated joins for small dimension tables
- REFRESH TABLE statement can be used to populate the MQT table with data
- Only REFRESH DEFERRED is supported
- CURRENT REFRESH AGE will need to be ANY for the optimizer to utilize the replicated MQT to process a query
- The ORGANIZE BY COLUMN option is required even if the default table organization (DFT_TABLE_ORG) is set to COLUMN

DB2 BLU MPP support © Copyright IBM Corporation 2017

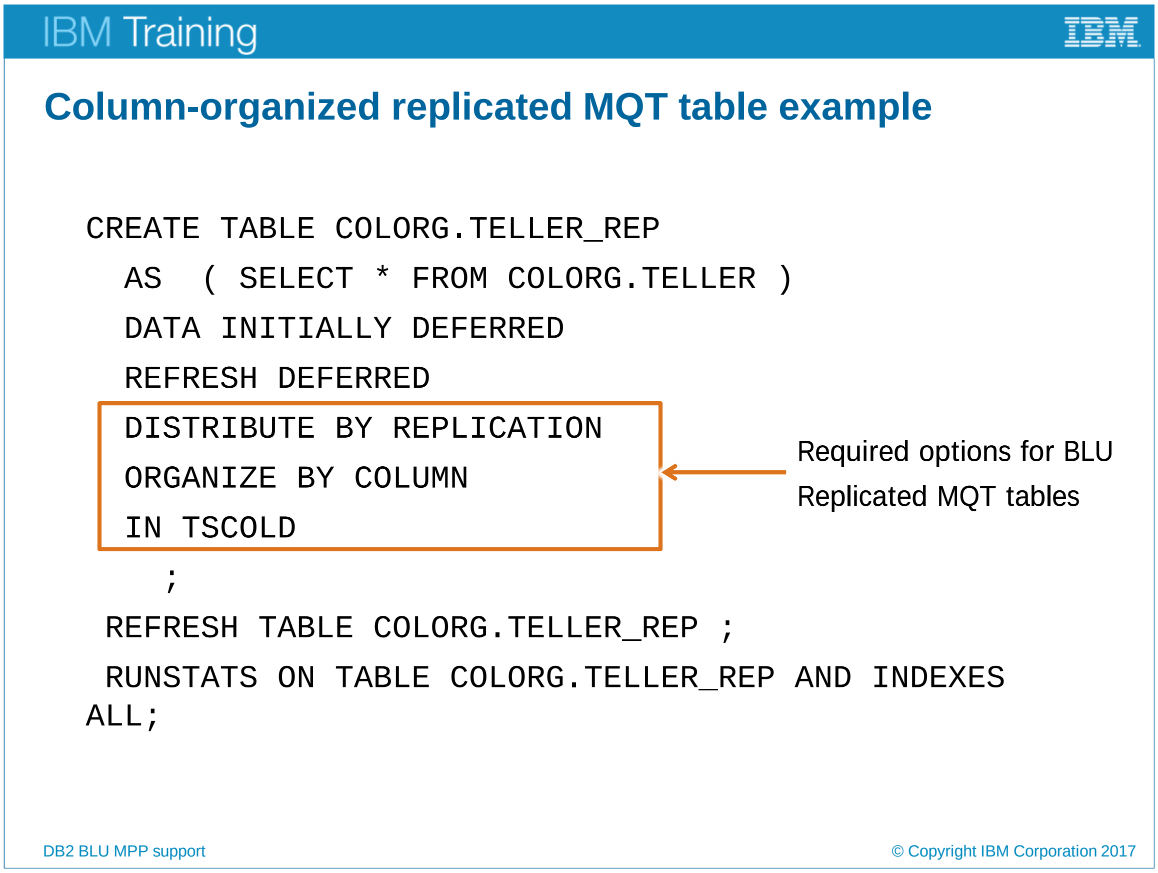
DB2 11.1 support for column-organized replicated MQT tables

DB2 11.1 now supports column-organized replicated Materialized Query Tables. With the addition of DB2 MPP database support in DB2 11.1, customers moving from row-organized tables to column-organized tables may want to convert some replicated MQT tables that are currently used for applications to column-organized.

The options REFRESH DEFERRED and ORGANIZE BY COLUMN will need to be included when a column-organized replicated MQT is created.

The REFRESH TABLE statement can be used to populate the replicated MQT contents.

The CURRENT REFRESH AGE will need to be set to ANY for the DB2 optimizer to consider use of REFRESH DEFERRED MQT tables in the access plan.



IBM Training

Column-organized replicated MQT table example

```
CREATE TABLE COLORG.TELLER_REP
  AS  ( SELECT * FROM COLORG.TELLER )
  DATA INITIALLY DEFERRED
  REFRESH DEFERRED
  DISTRIBUTE BY REPLICATION
  ORGANIZE BY COLUMN
  IN TSCOLD
    ;
 REFRESH TABLE COLORG.TELLER_REP ;
 RUNSTATS ON TABLE COLORG.TELLER_REP AND INDEXES
ALL;
```

Required options for BLU Replicated MQT tables

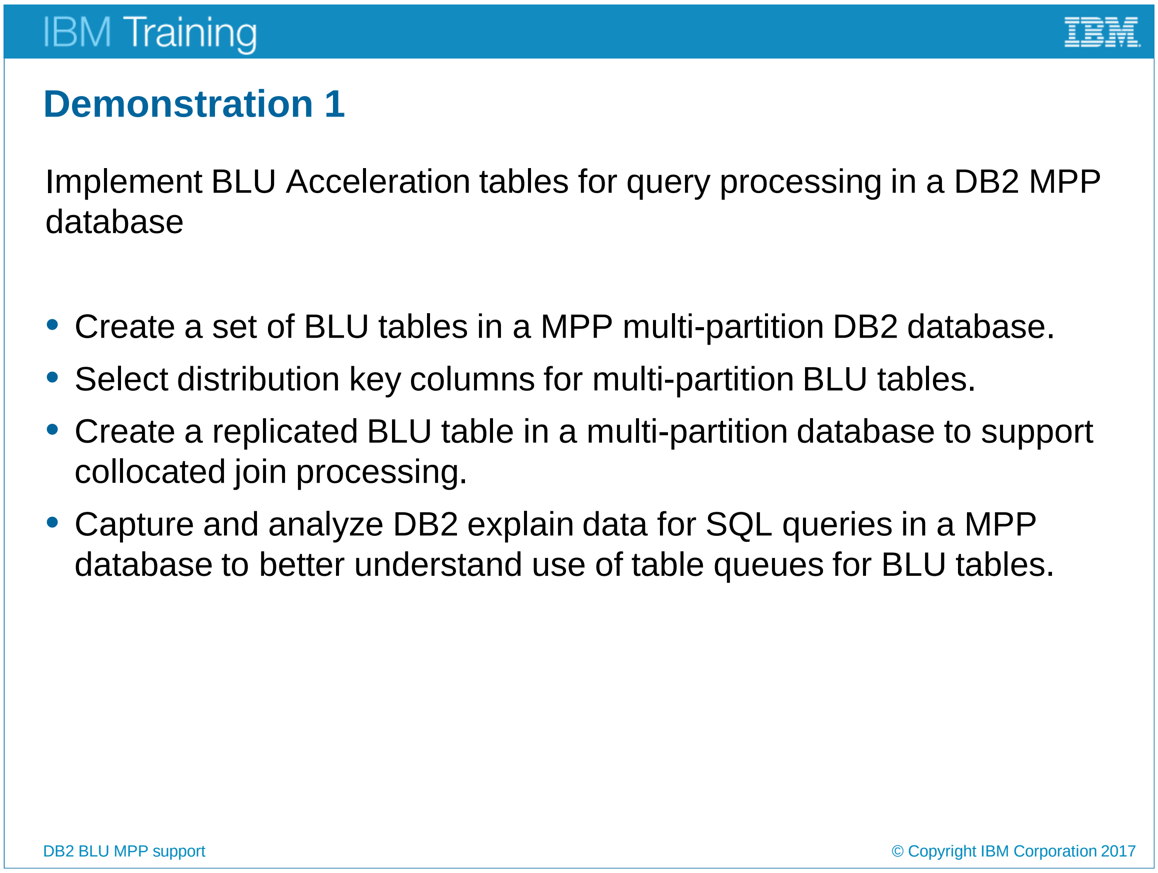
DB2 BLU MPP support

© Copyright IBM Corporation 2017

Column-organized replicated MQT table example

The slide shows a sample CREATE TABLE statement that would create a column-organized replicated MQT table.

The REFRESH TABLE and RUNSTATS statements would be used to populate the MQT with the results of the SELECT statement and collect table statistics.



IBM Training

Demonstration 1

Implement BLU Acceleration tables for query processing in a DB2 MPP database

- Create a set of BLU tables in a MPP multi-partition DB2 database.
- Select distribution key columns for multi-partition BLU tables.
- Create a replicated BLU table in a multi-partition database to support collocated join processing.
- Capture and analyze DB2 explain data for SQL queries in a MPP database to better understand use of table queues for BLU tables.

DB2 BLU MPP support

© Copyright IBM Corporation 2017

Demonstration 1

Demonstration 1: Implement BLU Acceleration tables for query processing in a DB2 MPP database

Purpose:
This demonstration will show you how to define and manage column-organized tables in a DB2 multi-partition MPP database. A set of column-organized tables will be created and loaded using data from an existing single partition DB2 database. A set of queries will be executed and analyzed to study the processing of BLU tables in a multi-partition database. The DB2 explain tool will be used to verify join processing and use of table queues to move data between partitions.

Task 1. Implement a set of BLU tables in a DB2 MPP database.

A three partition DB2 database named **BLUMPP** has been created in a DB2 instance owned by the Linux user **instmpp**. The partition numbers are 0, 1 and 2.

Several new database partition groups were defined.

- **smallgrp** - was defined for use with small tables and only contains partition 0.
- **biggrp** - was defined for larger tables and contains partitions 1 and 2.
- The default partition group contains all three partitions, 0, 1 and 2.

You will need to logout from the Linux system and login using the user **instmpp**, the instance owner for the multi-partition instance.

1. Logout from inst450 using the pulldown on the far right of the Redhat toolbar, then use the pulldown to the right of the user (inst450) icon -> **Log Out.** Press **Log Out** on the confirmation popup window.
2. Select **instmpp** on the Login screen and press Enter. Use the password provided by the instructor (else use ***ibm2blue***).

You will create a set of tables that will be loaded with data from an existing single partition database.

The file *mpptables.ddl* contains the SQL to create four new tables. The instance owner, **instmpp** will be used for the schema.

The file mpptables.ddl contains the following statements:

```
CREATE TABLE ACCT
      (ACCT_ID          INT              NOT NULL,
       NAME             CHAR(20)         NOT NULL,
       ACCT_GRP         SMALLINT         NOT NULL,
       BALANCE          DECIMAL(15,2)    NOT NULL,
       ADDRESS          CHAR(30)         NOT NULL,
       TEMP             CHAR(40)         NOT NULL)
       IN MPPacctd INDEX IN MPPACCTI
       DISTRIBUTE BY HASH (ACCT_ID) ;


CREATE TABLE BRANCH
      (BRANCH_ID        SMALLINT         NOT NULL,
       BRANCH_NAME      CHAR(20)         NOT NULL,
       BALANCE          DECIMAL(15,2)    NOT NULL,
       AREA_CODE        CHAR(4)          NOT NULL,
       ADDRESS          CHAR(30)         NOT NULL,
       TEMP             CHAR(40)         NOT NULL)
       IN MPPSMALL;

CREATE TABLE TELLER
      (TELLER_ID        SMALLINT         NOT NULL,
       TELLER_NAME      CHAR(20)         NOT NULL,
       BRANCH_ID        SMALLINT         NOT NULL,
       BALANCE          DECIMAL(15,2)    NOT NULL,
       TELLER_CODE      CHAR(2)          NOT NULL,
       ADDRESS          CHAR(30)         NOT NULL,
       TEMP             CHAR(40)         NOT NULL)
       IN MPPSMALL;

CREATE TABLE HISTORY
      (ACCT_ID          INTEGER          NOT NULL,
       TELLER_ID        SMALLINT         NOT NULL,
       BRANCH_ID        SMALLINT         NOT NULL,
       BALANCE          DECIMAL(15,2)    NOT NULL,
       DELTA            DECIMAL(9,2)     NOT NULL,
       PID              INTEGER          NOT NULL,
       TSTMP            TIMESTAMP        NOT NULL WITH DEFAULT,
       ACCTNAME         CHAR(20)         NOT NULL,
       TEMP             CHAR(6)          NOT NULL)
       IN USERSPACE1
       DISTRIBUTE BY RANDOM ;
```

The tables have the following characteristics:

- ACCT contains 1 million rows of account data. The table uses two table spaces that span partitions 1 and 2. The ACCT_ID column value will be used to distribute rows between the two partitions.
- BRANCH contains 100 rows of bank branch information. This table only uses partition 0 for storage.
- TELLER - contains 1000 rows of bank teller information. This table only uses partition 0 for storage.
- HISTORY - contains just under 500 thousand rows of bank transaction data. The table space used, USERSPACE1 spans the three database partitions. The DISTRIBUTE BY RANDOM clause causes rows to be randomly assigned to partitions, not based on any column value.

Connect to the BLUMPP database and use the file *checkcfg.sql* to show some of the database configuration options.

You will use the Linux Terminal session to enter DB2 commands.

3. Right-click the empty Linux desktop and select **Open in Terminal**.
4. Enter the following commands in the Linux terminal session:
   - **cd $HOME/blumpp**
   - **db2start** (this may take a few minutes. You will see 3 DB2START successful messages, one per DB partition)
   - **db2 connect to blumpp**
   - **db2 -tvf checkcfg.sql**

The query result will be similar to the following:

```
select dbpartitionnum, name as CFG_option, varchar(value,30)  as configured_value
from sysibmadm.dbcfg where name in
('sheapthres_shr','sortheap','util_heap_sz','dft_table_org','self_tuning_mem','num
_ioservers') order by  name, dbpartitionnum

DBPARTITIONNUM CFG_OPTION                       CONFIGURED_VALUE
-------------- -------------------------------- ------------------------------
             0 dft_table_org                    COLUMN
             0 num_ioservers                    4
             0 self_tuning_mem                  OFF
             0 sheapthres_shr                   32768
             0 sortheap                         32768
             0 util_heap_sz                     9660

  6 record(s) selected.
```

Use the file *mpptables.ddl* to create the four test tables.

5. In the terminal session, enter the following command:
   - **db2 -tvf mpptables.ddl**

   You will use the file *loadacct.sql* to execute a LOAD utility that accesses the ACCT table in the BLUTRAN database to load the ACCT table. The file contains the following statements:

```
declare rowacct cursor database blutran user inst450 using ibm2blue
for
   select * from roworg.acct ;
load from rowacct of cursor messages loadacctc.msg replace into
instmpp.acct ;
```

6. In the terminal session, enter the following command:
   - **db2 -tvf loadacct.sql**

   The load processing produces a set of message files with a prefix of loadacctc.msg.

   Review the message file that shows how data rows were partitioned for loading.

7. In the terminal session, enter the following command:
   - **cat loadacctc.msg.part.***

   The output will be similar to the following:

```
SQL27903I  "PARTITION" has started on partition "1" at time
"05/23/2017
10:53:06.411338".

SQL27950I  The type of the input data file is "3".

SQL27914I  The mode of operation is "PARTITION".

SQL27920I  This utility is using " 1" partitioning keys.

SQL27921I  "ACCT_ID"  Start:"0"  Len:"4"  Position:"1"  Type:"1".

SQL27935I  "PARTITION" has ended on partition "1" at time
"05/23/2017
10:53:17.173745".

SQL27936I  Elapsed time: " 0" hours, " 0" minutes, "11" seconds.

SQL27937I  Throughput: "90909" records/sec.

SQL27939I  Record counts for output partitions: partition number
"1". Record
count: "499946".

SQL27939I  Record counts for output partitions: partition number
"2". Record
count: "500054".
```

   The messages indicate the one million data rows were evenly divided between the two database partitions.

   You will use the file *loadhistory.sql* to execute a LOAD utility that accesses the HISTORY table in the BLUTRAN database to load the HISTORY table. The file contains the following statements:

```
declare rowhist cursor database blutran user inst450 using ibm2blue
for
   select * from roworg.history ;
load from rowhist of cursor messages loadhistc.msg replace into
instmpp.history ;
```

8. In the terminal session, enter the following command:

   - **db2 -tvf loadhistory.sql**

   The load processing produces a set of message files with a prefix of loadhistc.msg.

   The HISTORY table was create using the DISTRIBUTE BY RANDOM clause, so no partitioning message file is produced during loading. Three load messages files were produced during load processing.

   Review the message file that shows how data rows were loaded for partition 1.

9. In the terminal session, enter the following command:

   - **cat loadhistc.msg.load.001**

   The output will be similar to the following:

```
SQL3501W  The table space(s) in which the table resides will not be
placed in
backup pending state since forward recovery is disabled for the
database.

SQL3253N  The utility is beginning to load data from the SQL
statement "
select * from roworg.history " in database "blutran".

SQL3500W  The utility is beginning the "ANALYZE" phase at time
"05/23/2017
10:56:30.401297".

SQL3519W  Begin Load Consistency Point. Input record count = "0".

SQL3520W  Load Consistency Point was successful.

SQL3515W  The utility has finished the "ANALYZE" phase at time
"05/23/2017
10:56:40.224790".

SQL3500W  The utility is beginning the "LOAD" phase at time
"05/23/2017
10:56:40.225374".

SQL3110N  The utility has completed processing.  "163650" rows were
read from
the input file.

SQL3519W  Begin Load Consistency Point. Input record count =
"163650".

SQL3520W  Load Consistency Point was successful.
```

```
SQL3515W  The utility has finished the "LOAD" phase at time
"05/23/2017
10:56:42.534024".

SQL3500W  The utility is beginning the "BUILD" phase at time
"05/23/2017
10:56:42.538701".

SQL3213I  The indexing mode is "REBUILD".

SQL3515W  The utility has finished the "BUILD" phase at time
"05/23/2017
10:56:42.708412".
```

You will use the file *loadbranch.sql* to execute a LOAD utility that accesses the BRANCH table in the BLUTRAN database to load the BRANCH table.

10. In the terminal session, enter the following command:

    - **db2 -tvf loadbranch.sql**

    The output will be similar to the following:

```
declare rowbranch cursor database blutran user inst450 using
for select * from roworg.branch
DB20000I  The SQL command completed successfully.

load from rowbranch of cursor messages loadacctc.msg replace into
instmpp.branch

  Agent Type     Node     SQL Code     Result
__________________________________________________________________
__________
  LOAD           000      +00000000    Success.
__________________________________________________________________
__________
  PRE_PARTITION  000      +00000000    Success.
__________________________________________________________________
__________
  FILE_TRANSFER  000      +00000000    Success.
__________________________________________________________________
__________
  RESULTS:       1 of 1 LOADs completed successfully.
__________________________________________________________________
__________

Summary of LOAD Agents:
Number of rows read         = 100
Number of rows skipped      = 0
Number of rows loaded       = 100
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 100
```

You will use the file *loadteller.sql* to execute a LOAD utility that accesses the TELLER table in the BLUTRAN database to load the TELLER table. Only one partition is used for load processing.

11. In the terminal session, enter the following command:
    - **db2 -tvf loadteller.sql**

    The output will be similar to the following:

```
declare rowtel cursor database blutran user inst450 using
for select * from roworg.teller
DB20000I  The SQL command completed successfully.

load from rowtel of cursor messages loadtlr.msg replace into
instmpp.teller

  Agent Type     Node     SQL Code     Result
____________________________________________________________________
__________
  LOAD           000      +00000000    Success.
____________________________________________________________________
__________
  PRE_PARTITION  000      +00000000    Success.
____________________________________________________________________
__________
  FILE_TRANSFER  000      +00000000    Success.
____________________________________________________________________
__________
  RESULTS:       1 of 1 LOADs completed successfully.
____________________________________________________________________
__________

Summary of LOAD Agents:
Number of rows read         = 1000
Number of rows skipped      = 0
Number of rows loaded       = 1000
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 1000
```

The 1000 TELLER table rows are loaded to partition 0.

Use several SQL queries to show the storage required to support these four column-organized tables in the three partition database.

12. In the terminal session, enter the following command:

    - **db2 -tvf qsyscat.sql**

    The output will be similar to the following:

```
select varchar(tabschema,12) as schema, varchar(tabname,12) as
table, card, tableorg, npages, fpages, mpages, pctpagessaved from
syscat.tables where tabschema = 'INSTMPP' order by tabname

SCHEMA       TABLE        CARD                 TABLEORG NPAGES
FPAGES               MPAGES               PCTPAGESSAVED
------------ ------------ -------------------- -------- ------------
-------- -------------------- -------------------- -------------
INSTMPP      ACCT                       999892 C
142                  144                    1            96
INSTMPP      BRANCH                        100 C
8                    9                    1            0
INSTMPP      HISTORY                    491400 C
225                  255                    10           77
INSTMPP      TELLER                       1000 C
9                   10                    1            0

  4 record(s) selected.
```

The PCTPAGESSAVED for the larger tables, ACCT and HISTORY show high compression results. The two small tables require storage for each column and the small number of rows do not show any space savings from compression.

13. In the terminal session, enter the following command:
    - **db2 -tvf tabinfocol.sql**

    The output will be similar to the following:

```
select SUBSTR(TABSCHEMA,1,10) AS SCHEMA , SUBSTR(TABNAME,1,12) AS
TABLE , dbpartitionnum, DATA_OBJECT_P_SIZE, INDEX_OBJECT_P_SIZE ,
COL_OBJECT_P_SIZE from table ( admin_get_tab_info (NULL,NULL  ) ) AS
TABINFO where tabschema in ('INSTMPP') order by
TABNAME,dbpartitionnum

SCHEMA     TABLE        DBPARTITIONNUM DATA_OBJECT_P_SIZE
INDEX_OBJECT_P_SIZE  COL_OBJECT_P_SIZE
---------- ------------ -------------- -------------------- --------
----------- -------------------
INSTMPP    ACCT                      1                  256
256                 3072
INSTMPP    ACCT                      2                  256
256                 3072
INSTMPP    BRANCH                    0                  256
256                 1152
INSTMPP    HISTORY                   0                  512
256                 3200
INSTMPP    HISTORY                   1                  512
256                 3200
INSTMPP    HISTORY                   2                  512
256                 3200
INSTMPP    TELLER                    0                  256
256                 1280

  7 record(s) selected.
```

    The query result shows the storage required for each table component, data, index and column organized, for each table on the three database partitions.

    For these column-organized tables, most of the storage is allocated for the column-organized component.

    Use the SQL file *pkeys.ddl* to define a set of NOT ENFORCED primary key and foreign key constraints on these four tables. These will help the DB2 optimizer to understand table data relationships when these tables are joined without the overhead of checking the constraints. The file contains the following statements:

```
alter table instmpp.acct add primary key (acct_id) not enforced ;
alter table instmpp.branch add primary key (branch_id)  not enforced
;
alter table instmpp.teller add primary key (teller_id)  not enforced
;

alter table instmpp.history add constraint fkeyteller
```

```
      foreign key (teller_id) references instmpp.teller not enforced
;

alter table instmpp.history add constraint fkeybranch
      foreign key (branch_id) references instmpp.branch not enforced
;

alter table instmpp.history add constraint fkeyacct
      foreign key (acct_id) references instmpp.acct not enforced ;
```

14. In the terminal session, enter the following command:
    - **db2 -tvf pkeys.ddl**

Task 2. Run a set of SQL queries using the BLU tables in the partitioned database and analyze the access plans.

You will use db2batch to execute several SQL queries and collect the statistics that show processing for each database partition. You will review the access plans to note the use of table queues to move data between partitions to produce the results.

The first SQL statement will access a single table, HISTORY and produce a summarized result.

The file *mppquery1.sql* contains the following SQL statements:

```
set current explain mode yes;
SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance,
count(*) as br_trans
   FROM instmpp.HISTORY AS HISTORY
   where branch_id between 10 and 30 and teller_id > 800
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;
set current explain mode no;

--#COMMENT CHECK STATS
SELECT member as partition, total_wait_time, pool_read_time,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as
total_l_reads
   from table(mon_get_connection(null,-2)) as con
  where application_name = 'db2batch'
```

You will use the Linux Terminal session to enter DB2 commands.

1. Enter the following commands in the Linux terminal session:
   - **cd $HOME/blumpp**
   - **db2batch -d blumpp -f mppquery1.sql -iso cs | tee bat_mppq1.txt**
   - **more bat_mppq1.txt**

The db2batch run also included a SQL query that shows processing for each of the three database partitions.

```
SELECT member as partition, total_wait_time, pool_read_time,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-2)) as con
  where application_name = 'db2batch'
 ;

PARTITION TOTAL_WAIT_TIME      POOL_READ_TIME       TOTAL_L_READS
--------- -------------------- -------------------- --------------------
        0                16948                 1183                  626
        1                 8249                  419                  199
        2                 8893                  963                  199
```

The HISTORY table was defined to utilize all three partitions. The sample result shows more processing on partition 0, the coordinator partition, which performs the SQL compilation and the work to send the final result to the application.

Use db2exfmt to format the access plan for the SQL query.

2. Enter the following commands in the Linux terminal session:

- **db2exfmt -1 -d blumpp -o exp_mppq1.txt**
- **more exp_mppq1.txt**

Review the access plan section of the explain report.

For example:

```
Access Plan:
-----------
    Total Cost:           177.387
    Query Degree:         1


         Rows
        RETURN
        (   1)
         Cost
          I/O
           |
        66.8538
        CTQ
        (   2)
        177.387
          22
           |
        66.8538
        MDTQ
        (   3)
        177.386
          22
```

```
             |
           22.2846
           TBSCAN
           (   4)
           177.361
             22
             |
           22.2846
           SORT
           (   5)
           177.36
             22
             |
           22.2846
           GRPBY
           (   6)
           177.35
             22
             |
           66.8538
           DTQ
           (   7)
           177.348
             22
             |
           66.8538
           GRPBY
           (   8)
           177.309
             22
             |
           23856.2
           TBSCAN
           (   9)
           176.879
             22
             |
           163800
     CO-TABLE: INSTMPP
           HISTORY
             Q1
```

The processing flow is from the bottom to the top. You can review the detailed information in the explain report for each processing step.

Here is a summary of the processing:

- Step 9 is a table scan of the HISTORY table that processes on all three partitions. The count 163800 is the row count from one database partition. This scan applies several predicates and reduces the result to an estimated 23856 per partition.
- Step 8 is a group by operation on the BRANCH_ID column, resulting in an estimated 66 rows.
- Step 7 is table queue used to send the result rows to the three partitions hashed by the BRANCH_ID value.
- Step 6 is a group by operation that combines the partially grouped results read from the previous table queue.
- Step 5 is a sort operation to order the results by the BRANCH_ID value on the three partitions.
- Step 4 scans the sorted result from the previous step
- Step 3 is a table queue used to send the results from the three partitions to the coordinator partition.
- Step 2 is the column-organized table queue that puts the results into row format.

The next SQL statement will join two tables, HISTORY and TELLER and produce a summarized result. The HISTORY table spans three database partitions but the TELLER table only uses partition 0.

The file *mppquery2.sql* contains the following SQL statements:

```
set current explain mode yes;

SELECT HISTORY.TELLER_ID, sum(HISTORY.BALANCE) as total_balance,
TELLER.TELLER_NAME , count(*) as transactions
  FROM instmpp.HISTORY AS HISTORY, instmpp.TELLER AS TELLER
   WHERE HISTORY.TELLER_ID = TELLER.TELLER_ID
    and HISTORY.teller_id between 10 and 100
   GROUP BY HISTORY.TELLER_ID, TELLER.TELLER_NAME ;

set current explain mode no;

--#COMMENT CHECK STATS
SELECT member as partition, total_wait_time, pool_read_time,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as
total_l_reads
   from table(mon_get_connection(null,-2)) as con
  where application_name = 'db2batch'
```

3. Enter the following commands in the Linux terminal session:
   - **db2batch -d blumpp -f mppquery2.sql -iso cs | tee bat_mppq2.txt**
   - **more bat_mppq2.txt**

   The db2batch run also included a SQL query that shows processing for each of the three database partitions.

```
SELECT member as partition, total_wait_time, pool_read_time,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-2)) as con
  where application_name = 'db2batch'
 ;

PARTITION TOTAL_WAIT_TIME      POOL_READ_TIME       TOTAL_L_READS
--------- -------------------- -------------------- --------------------
        0                  653                  131                  414
        1                   44                   12                   54
        2                   39                   12                   54
```

   The HISTORY table was defined to utilize all three partitions. The sample result shows more processing on partition 0, the coordinator partition, which performs the SQL compilation and the work to send the final result to the application.

   Use db2exfmt to format the access plan for the SQL query.

4. Enter the following commands in the Linux terminal session:
   - **db2exfmt -1 -d blumpp -o exp_mppq2.txt**
   - **more exp_mppq2.txt**

   Review the access plan section of the explain report.

   For example:

```
Access Plan:
-----------
    Total Cost:          188.992
    Query Degree:        1


                Rows
               RETURN
               (   1)
                Cost
                 I/O
                 |
               99.0196
               CTQ
               (   2)
               188.992
                 24
                 |
               99.0196
               DTQ
               (   3)
               188.99
                 24
                 |
               99.0196
               ^HSJOIN
               (   4)
               188.953
                 24
            /----+----\
         1000         98.8216
        TBSCAN        GRPBY
        (   5)        (   6)
        21.1889       167.751
           3            21
          |             |
         1000         296.465
  CO-TABLE: INSTMPP   DTQ
        TELLER        (   7)
          Q1          167.744
                        21
                        |
                      98.8216
                      GRPBY
                      (   8)
                      167.682
                        21
                        |
                      16219.4
                      TBSCAN
                      (   9)
                      167.388
                        21
                        |
                      163800
                CO-TABLE: INSTMPP
                      HISTORY
                        Q2
```

The hash join operation, step 4, would be described as a 'Directed Inner' join, because the inner table HISTORY was previously sent from its three partitions (step 7) to be joined on the one partition of the TELLER table using a table queue.

Look at the various table queue operations used to produce the SQL result.

You can review the detailed information in the explain report for each processing step.

Here is a summary of the processing:

- Step 7 this table queue is used to send HISTORY table grouped results from the three partitions used for this table to the single partition where the TELLER table is stored, partition 0.
- Step 3 this table queue is used to send the joined HISTORY and TELLER data to the coordinator partition.
- Step 2 is the column-organized table queue that puts the results into row format.

The next SQL statement will join three tables, ACCT, HISTORY and TELLER. The ACCT table spans partitions 1 and 2, the HISTORY table spans the three database partitions but the TELLER table only uses partition 0.

The file *mppquery3.sql* contains the following SQL statements:

```
set current explain mode yes;

SELECT ACCT.ACCT_ID, ACCT.NAME, TELLER.TELLER_ID, TELLER.TELLER_NAME,
  TELLER.TELLER_CODE, ACCT.ADDRESS ,
  HISTORY.BRANCH_ID, HISTORY.BALANCE, HISTORY.PID, HISTORY.TEMP
   FROM instmpp.ACCT AS ACCT, instmpp.TELLER AS TELLER,
instmpp.HISTORY AS HISTORY
   WHERE ACCT.ACCT_ID = HISTORY.ACCT_ID
   AND ACCT.ACCT_GRP BETWEEN 100 AND 200
   AND HISTORY.TELLER_ID = TELLER.TELLER_ID
   AND HISTORY.BRANCH_ID BETWEEN 40 AND 50
   ORDER BY HISTORY.PID ASC ;

set current explain mode no;

--#COMMENT CHECK STATS
SELECT member as partition, total_wait_time, pool_read_time,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as
total_l_reads
   from table(mon_get_connection(null,-2)) as con
  where application_name = 'db2batch'
```

5. Enter the following commands in the Linux terminal session:
   - **db2batch -d blumpp -f mppquery3.sql -iso cs | tee bat_mppq3.txt**
   - **more bat_mppq3.txt**

The db2batch run also included a SQL query that shows processing for each of the three database partitions.

```
SELECT member as partition, total_wait_time, pool_read_time,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-2)) as con
  where application_name = 'db2batch'
 ;

PARTITION TOTAL_WAIT_TIME      POOL_READ_TIME       TOTAL_L_READS
--------- -------------------- -------------------- --------------------
        0                 4450                  176                  410
        1                 1752                  668                  440
        2                 1916                  962                  437
```

The HISTORY table was defined to utilize all three partitions. The large ACCT table is stored on partitions 1 and 2. The SQL processing is divided across the three partitions.

Use db2exfmt to format the access plan for the SQL query.

6. Enter the following commands in the Linux terminal session:

- **db2exfmt -1 -d blumpp -o exp_mppq3.txt**
- **more exp_mppq3.txt**

Review the access plan section of the explain report.

For example:

```
Access Plan:
-----------
    Total Cost:          267.4
    Query Degree:        1


                     Rows
                    RETURN
                    (   1)
                     Cost
                      I/O
                      |
                    2226.72
                    CTQ
                    (   2)
                     267.4
                      81
                      |
                    2226.72
                    MDTQ
                    (   3)
                    267.364
                      81
```

```
                          |
                        1113.36
                        TBSCAN
                        (   4)
                        266.635
                          81
                          |
                        1113.36
                        SORT
                        (   5)
                        266.593
                          81
                          |
                        1113.36
                        HSJOIN
                        (   6)
                        265.883
                          81
                 /----------+----------\
              1000                        1111.13
             BTQ                          HSJOIN
             (   7)                       (   9)
             21.4611                      244.389
                3                           78
               |                        /----+----\
              1000                  53395.8       10403.5
             TBSCAN                 TBSCAN        DTQ
             (   8)                 (  10)        (  11)
             21.1889                147.456       96.5043
                3                     42            36
               |                     |             |
              1000                  499946        6935.69
       CO-TABLE: INSTMPP   CO-TABLE: INSTMPP   TBSCAN
             TELLER                ACCT           (  12)
               Q2                   Q3            92.9816
                                                    36
                                                   |
                                                  163800
                                            CO-TABLE: INSTMPP
                                                  HISTORY
                                                    Q1
```

The hash join, step 9 joins the ACCT and HISTORY tables using the ACCT_ID column. This is a directed-inner join. Since the HISTORY table was randomly distributed, the HISTORY data produced by the table scan, step 12 must be sent to be joined with the ACCT table using the directed table queue, step 11.

The second hash join, step 6, joins the TELLER table with the composite ACCT and HISTORY data. This is a broadcast outer join, since the TELLER table data is broadcast to the two partitions where ACCT data is stored for join processing.

The join between the ACCT and HISTORY tables is based on the ACCT_ID column.

The ACCT table data is distributed on this column value. If the HISTORY table was also distributed using the same column and stored using the same partition group, these two tables could perform collocated joins.

You can use the SQL statements in the file *mpphist2.ddl* to drop and recreate the HISTORY table to enable collocated joins. The file *mpphist2.ddl* contains the following SQL text:

```
DROP TABLE HISTORY;
CREATE TABLE HISTORY
      (ACCT_ID          INTEGER         NOT NULL,
       TELLER_ID        SMALLINT        NOT NULL,
       BRANCH_ID        SMALLINT        NOT NULL,
       BALANCE          DECIMAL(15,2)   NOT NULL,
       DELTA            DECIMAL(9,2)    NOT NULL,
       PID              INTEGER         NOT NULL,
       TSTMP            TIMESTAMP       NOT NULL WITH DEFAULT,
       ACCTNAME         CHAR(20)        NOT NULL,
       TEMP             CHAR(6)         NOT NULL)
       IN MPPhist
       DISTRIBUTE BY HASH (ACCT_ID) ;

alter table instmpp.history add constraint fkeyteller
      foreign key (teller_id) references instmpp.teller not enforced
;

alter table instmpp.history add constraint fkeybranch
      foreign key (branch_id) references instmpp.branch not enforced
;

alter table instmpp.history add constraint fkeyacct
      foreign key (acct_id) references instmpp.acct not enforced ;
```

7. Enter the following commands in the Linux terminal session:

   - **db2 connect to blumpp**
   - **db2 -tvf mpphist2.ddl**

   The HISTORY table can be reloaded with data using the file *loadhist2.sql*.

8. Enter the following commands in the Linux terminal session:
   - **db2 connect to blumpp**
   - **db2 -tvf loadhist2.sql**

   The output will be similar to the following:

```
declare rowhist cursor database blutran user inst450 using          for select *
from roworg.history
DB20000I  The SQL command completed successfully.

load from rowhist of cursor messages loadhist2.msg replace into instmpp.history

  Agent Type     Node     SQL Code     Result
______________________________________________________________________________
  LOAD           001      +00000000    Success.
______________________________________________________________________________
  LOAD           002      +00000000    Success.
______________________________________________________________________________
  PARTITION      001      +00000000    Success.
______________________________________________________________________________
  RESULTS:       2 of 2 LOADs completed successfully.
______________________________________________________________________________

Summary of Partitioning Agents:
Rows Read                   = 490864
Rows Rejected               = 0
Rows Partitioned            = 490864

Summary of LOAD Agents:
Number of rows read         = 490864
Number of rows skipped      = 0
Number of rows loaded       = 490864
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 490864
```

   Use db2batch to rerun the three table join query using the newly defined HISTORY table.

9. Enter the following commands in the Linux terminal session:
   - **db2batch -d blumpp -f mppquery3.sql -iso cs | tee bat_mppq32.txt**
   - **more bat_mppq32.txt**

   The db2batch run also included a SQL query that shows processing for each of the three database partitions.

```
SELECT member as partition, total_wait_time, pool_read_time,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-2)) as con
  where application_name = 'db2batch'
 ;

PARTITION TOTAL_WAIT_TIME      POOL_READ_TIME       TOTAL_L_READS
--------- -------------------- -------------------- --------------------
        0                 2692                  294                  604
        1                 1661                  864                  490
        2                 1789                  789                  480
```

   The SQL processing is divided across the three partitions.

   Use db2exfmt to format the access plan for the SQL query.

10. Enter the following commands in the Linux terminal session:
    - **db2exfmt -1 -d blumpp -o exp_mppq32.txt**
    - **more exp_mppq32.txt**

    Review the access plan section of the explain report.

    For example:

```
Access Plan:
-----------
    Total Cost:          297.074
    Query Degree:        1


                           Rows
                          RETURN
                          (   1)
                           Cost
                            I/O
                            |
                          2183.86
                          CTQ
                          (   2)
                          297.074
                            90
                            |
                          2183.86
```

```
                     MDTQ
                     (   3)
                     297.038
                       90
                       |
                     1091.93
                     TBSCAN
                     (   4)
                     296.323
                       90
                       |
                     1091.93
                     SORT
                     (   5)
                     296.282
                       90
                       |
                     1091.93
                     HSJOIN
                     (   6)
                     295.587
                       90
          /------------+------------\
       1000                         1089.75
      BTQ                           HSJOIN
      (   7)                        (   9)
      21.4611                       274.093
         3                            87
        |                     /-------+-------\
       1000               53395.8             10203.3
      TBSCAN              TBSCAN              TBSCAN
      (   8)              (  10)              (  11)
      21.1889             147.456             126.212
         3                  42                  45
        |                   |                   |
       1000               499946              243307
CO-TABLE: INSTMPP   CO-TABLE: INSTMPP   CO-TABLE: INSTMPP
      TELLER               ACCT               HISTORY
        Q2                  Q3                  Q1
```

The hash join, step 9 joins the ACCT and HISTORY tables using the ACCT_ID column. This is a directed-inner join. With the redefined HOSTORY table. This is now a collocated join. No table queue is needed to join these two tables.

The second hash join, step 6, joins the TELLER table with the composite ACCT and HISTORY data. This is a broadcast outer join, since the TELLER table data is broadcast to the two partitions where ACCT data is stored for join processing.

The join between the ACCT and HISTORY tables is based on the ACCT_ID column.

We can define a replicated MQT for the TELLER table to provide a local copy on the two partitions where the other two tables are stored and avoid the need to use a table queue to relocate the TELLER data.

You can use the SQL statements in the file *teller_mqt.ddl* to drop and recreate the HISTORY table to enable collocated joins. The file *teller_mqt.ddl* contains the following SQL text:

```
DROP TABLE TELLER_REP ;
CREATE TABLE TELLER_REP AS (SELECT * FROM INSTMPP.TELLER )
 DATA INITIALLY DEFERRED
 REFRESH DEFERRED
 DISTRIBUTE BY REPLICATION
 ORGANIZE BY COLUMN
 IN MPPHIST ;
REFRESH TABLE TELLER_REP ;
RUNSTATS ON TABLE TELLER_REP AND INDEXES ALL;
```

11. Enter the following commands in the Linux terminal session:
    - **db2 connect to blumpp**
    - **db2 -tvf teller_mqt.ddl**

    Use db2batch to rerun the three table join query using the newly defined HISTORY table.
12. Enter the following commands in the Linux terminal session:
    - **db2batch -d blumpp -f mppquery3mqt.sql -iso cs | tee bat_mppq33.txt**

    Use db2exfmt to format the access plan for the SQL query.
13. Enter the following commands in the Linux terminal session:
    - **db2exfmt -1 -d blumpp -o exp_mppq33.txt**
    - **more exp_mppq33.txt**

    Review the access plan and diagnostic messages sections of the explain report.

    For example:

```
Access Plan:
-----------
	Total Cost: 		298.488
	Query Degree:		1


                              Rows
                             RETURN
                             (   1)
```

```
                  Cost
                   I/O
                   |
                 2183.86
                 DTQ
                 (   2)
                 298.488
                   90
                   |
                 2183.86
                 CTQ
                 (   3)
                 297.513
                   90
                   |
                 2183.86
                 TBSCAN
                 (   4)
                 297.477
                   90
                   |
                 2183.86
                 SORT
                 (   5)
                 297.395
                   90
                   |
                 2183.86
                 ^HSJOIN
                 (   6)
                 295.913
                   90
              /----+-----\
          2179.5          1000
          BTQ            TBSCAN
          (   7)         (  11)
          274.699        21.1889
            87              3
            |              |
          1089.75         1000
          HSJOIN   CO-TABLE: INSTMPP
          (   8)         TELLER
          274.093          Q2
            87
    /-------+-------\
53395.8             10203.3
TBSCAN              TBSCAN
(   9)              (  10)
147.456             126.212
```

```
         42                  45
         |                   |
       499946              243307
 CO-TABLE: INSTMPP   CO-TABLE: INSTMPP
        ACCT               HISTORY
         Q3                  Q1


Operator Symbols :
------------------

   Symbol      Description
   ---------   ------------------------------------------
   >JOIN     : Left outer join
    JOIN<    : Right outer join
   >JOIN<    : Full outer join
   xJOIN     : Left antijoin
    JOINx    : Right antijoin
   ^JOIN     : Left early out
    JOIN^    : Right early out
    ATQ      : Asynchrony
    BTQ      : Broadcast
    CTQ      : Column-organized data
    DTQ      : Directed
    LTQ      : Intra-partition parallelism
    MTQ      : Merging (sorted)
    STQ      : Scatter
   RCTQ      : Column-organized data with row as the source
    XTQ      : XML aggregation
     TQ*     : Listener


Extended Diagnostic Information:
--------------------------------

Diagnostic Identifier:  1
Diagnostic Details:     EXP0079W  The following MQT was not used in
the
              final access plan, because the plan cost with this
              MQT was more expensive or a better candidate was
              available: "INSTMPP "."TELLER_REP".
Diagnostic Identifier:  2
Diagnostic Details:     EXP0148W  The following MQT or statistical
view was
              considered in query matching: "INSTMPP ".
              "TELLER_REP".
```

The access plan in the sample report is slightly different from the previous plan but the diagnostic messages indicate the replicated MQT was considered but not selected, since another access plan had a lower estimated cost.

You can force the DB2 optimizer to include the replicated MQT in the access plan using optimization guidelines.

The file *mppquery3mqt_opt.sql* contains the same SQL query with some additional text to direct the use of the replicated MQT table. The file contains the following text:

```
--#COMMENT MPP QUERY 3 Join with optimizer guidelines
--
set current refresh age any ;
set current explain mode explain;

SELECT ACCT.ACCT_ID, ACCT.NAME, TELLER.TELLER_ID, TELLER.TELLER_NAME,
  TELLER.TELLER_CODE, ACCT.ADDRESS ,
  HISTORY.BRANCH_ID, HISTORY.BALANCE, HISTORY.PID, HISTORY.TEMP
   FROM instmpp.ACCT AS ACCT, instmpp.TELLER AS TELLER, instmpp.HISTORY AS HISTORY
   WHERE ACCT.ACCT_ID = HISTORY.ACCT_ID
   AND ACCT.ACCT_GRP BETWEEN 100 AND 200
   AND HISTORY.TELLER_ID = TELLER.TELLER_ID
   AND HISTORY.BRANCH_ID BETWEEN 40 AND 50
   ORDER BY HISTORY.PID ASC
/* <OPTGUIDELINES><MQTENFORCE NAME='INSTMPP.TELLER_REP'/></OPTGUIDELINES>*/
;

set current explain mode no;
```

Use the file *mppquery3mqt_opt.sql* to generate the access plan without executing the SQL statement.

14. Enter the following commands in the Linux terminal session:

- **db2 -tvf mppquery3mqt_opt.sql**
- **db2exfmt -1 -d blumpp -o exp_mppq34.txt**
- **more exp_mppq34.txt**

Review the access plan and diagnostic messages sections of the explain report.

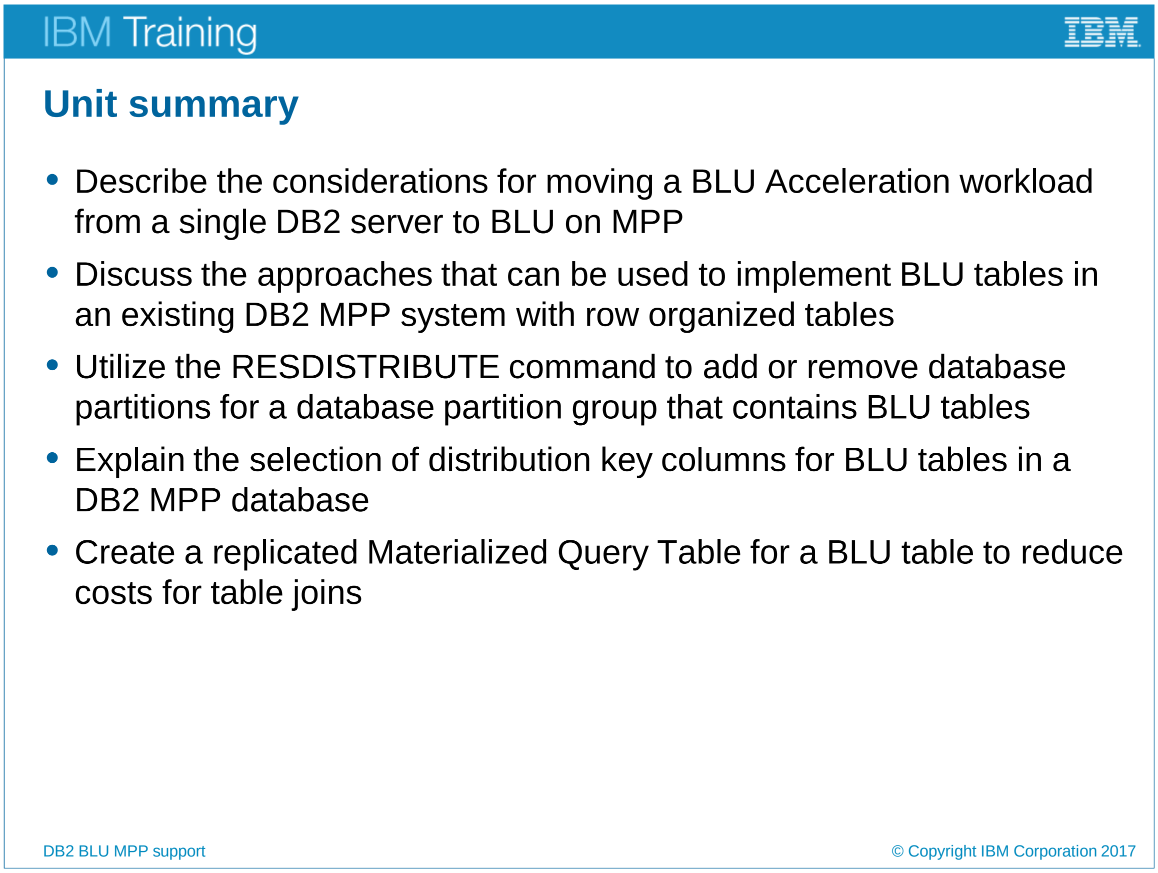
For example:

```
Access Plan:
-----------
    Total Cost:          353.165
    Query Degree:        1


                              Rows
                             RETURN
                             (   1)
                              Cost
```

```
                    I/O
                    |
                  2183.86
                  CTQ
                  (   2)
                  353.165
                    98
                    |
                  2183.86
                  MDTQ
                  (   3)
                  353.129
                    98
                    |
                  1091.93
                  TBSCAN
                  (   4)
                  352.414
                    98
                    |
                  1091.93
                  SORT
                  (   5)
                  352.373
                    98
                    |
                  1091.93
                  HSJOIN
                  (   6)
                  351.678
                    98
          /---------+---------\
       1000                    1089.75
      TBSCAN                   HSJOIN
      (   7)                   (   8)
      77.552                   274.093
        11                       87
        |               /-------+-------\
       1000         53395.8             10203.3
CO-TABLE: INSTMPP   TBSCAN              TBSCAN
    TELLER_REP      (   9)              (  10)
        Q2          147.456             126.212
                      42                  45
                      |                   |
                    499946              243307
              CO-TABLE: INSTMPP   CO-TABLE: INSTMPP
                     ACCT               HISTORY
                      Q3                  Q1
```

```
Extended Diagnostic Information:
--------------------------------

Diagnostic Identifier:   1
Diagnostic Details:      EXP0148W  The following MQT or statistical
view was
              considered in query matching: "INSTMPP ".
              "TELLER_REP".
Diagnostic Identifier:   2
Diagnostic Details:      EXP0149W  The following MQT was used (from
those
              considered) in query matching: "INSTMPP ".
              "TELLER_REP".
```

The access plan now has the three table joins using collocated joins without the additional table queues. The estimated cost for this plan is slightly higher than the one selected by the optimizer when no optimization guideline was included.



IBM Training

Unit summary

- Describe the considerations for moving a BLU Acceleration workload from a single DB2 server to BLU on MPP
- Discuss the approaches that can be used to implement BLU tables in an existing DB2 MPP system with row organized tables
- Utilize the RESDISTRIBUTE command to add or remove database partitions for a database partition group that contains BLU tables
- Explain the selection of distribution key columns for BLU tables in a DB2 MPP database
- Create a replicated Materialized Query Table for a BLU table to reduce costs for table joins

DB2 BLU MPP support

Unit summary

IBM Training