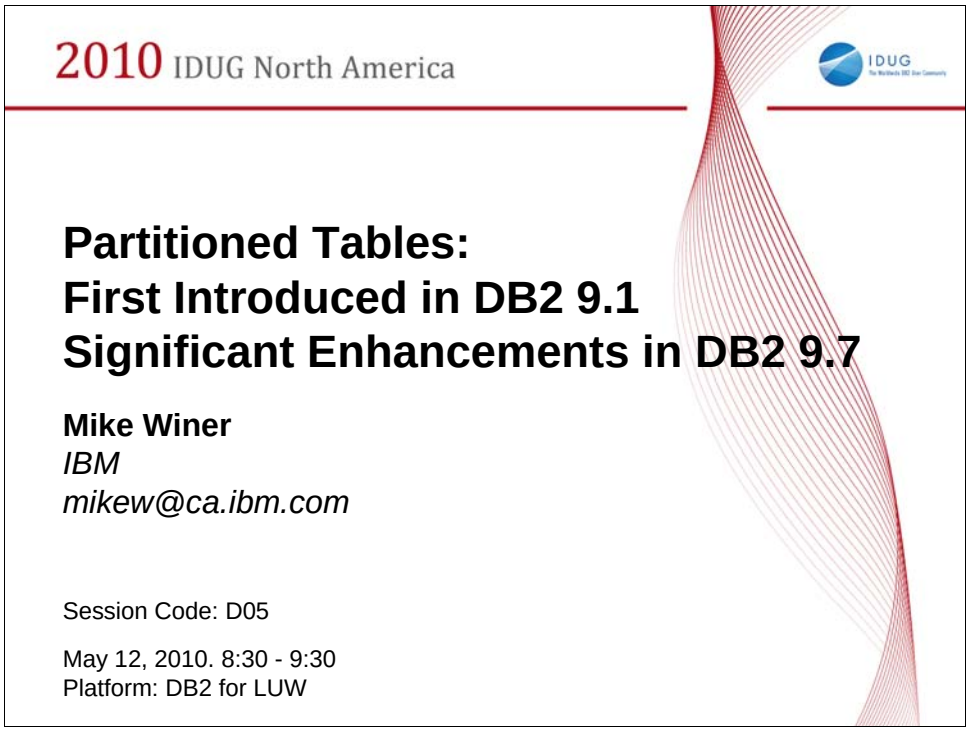


Presentation Abstract

This session will cover the significant enhancements to Partitioned Tables in DB2 for LUW since they were first introduced in DB2 9.1. To begin with, an overview of the partitioned table architecture, example use cases, ease of use using DDL, and best practices will be presented.

New to DB2 9.7, partitioned indexes are now supported, allowing each data partition to be indexed separately compared to the non-partitioned indexes available in prior versions. This session will cover partitioned indexes and how they can improve and increase usability, scalability, performance, etc.

In DB2 9.7 FP1, performance and availability on partitioned tables are improved. DETACH of a data partition no longer requires all current access to the table to drain, allowing long running queries to continue without impeding the DETACH operation. In addition, DB2 9.7 FP1 will also allow users to issue REORG operations on a single data partition vs. the entire partitioned table, allowing for faster REORG operations, increasing the availability to other data partitions of the partitioned table.

We will share some ideas on a few additional enhancements that are on our white-boards, no commitments of course! The audience will have their own ideas of other things they might like to see from DB2 for partitioned tables. Time permitting, an open discussion will be held.

The agenda of the session.

First up is a description of what things DB2 locks, how locks are physically represented, and which configuration parameters are used to manage lock memory.

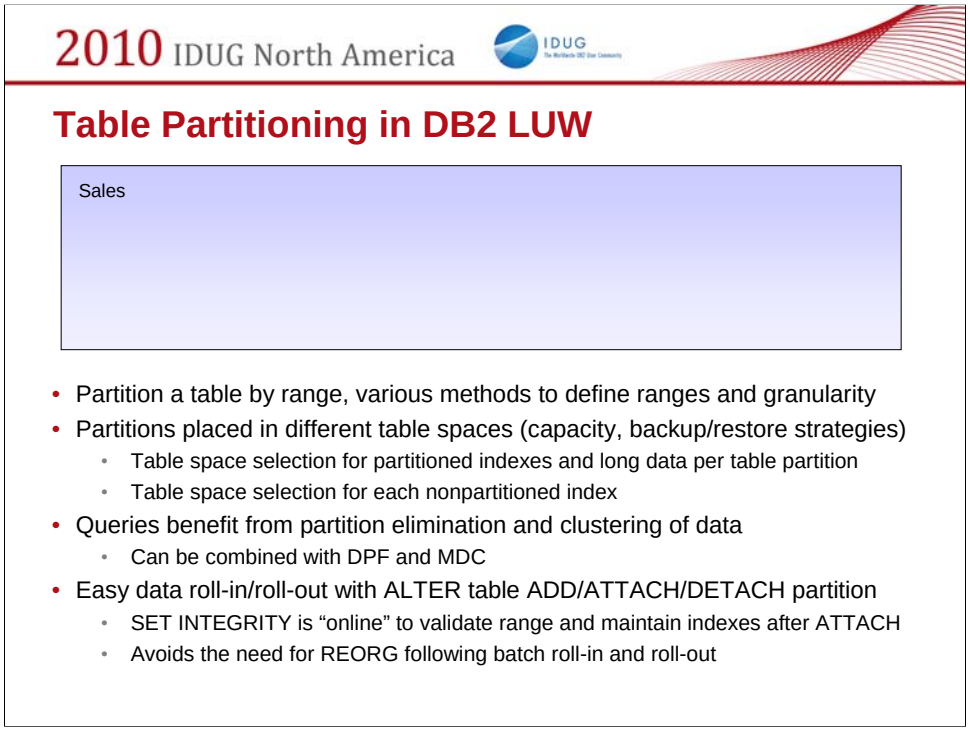


Table Partitioning allows you to create a table where each range of the data in the table is stored separately. For example, you can partition a table by month. Now all data for a given month will be kept together. In fact, internally, the database represents each range as a separate “table”.

There are many advantages:

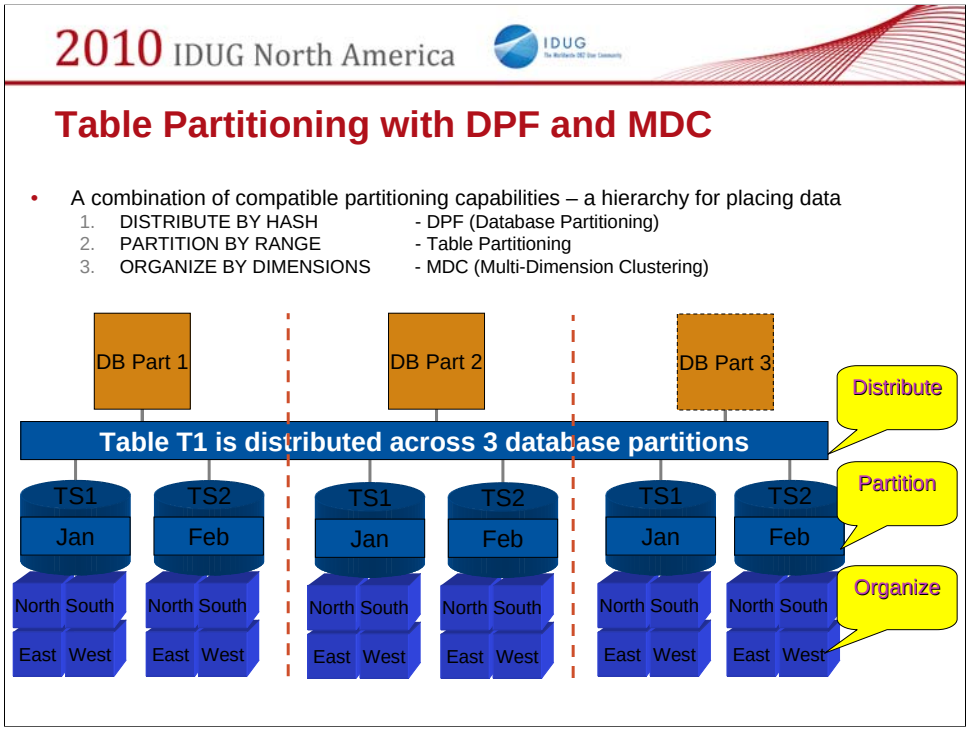
-Improved data roll-out via ALTER TABLE …DETACH PARTITION… to roll-out a range by transforming a range into an independent table instantly

-Improved data roll-in via ALTER TABLE …ATTACH PARTITION … to roll-in large amount of data by incorporating an independent table into an existing partitioned table without data copying or moving.

-Improved query performance via partition elimination. The database will scan only those ranges that are relevant to the query based on the query predicate. We call this "partition elimination“, and it’s determined not only during compilation by the optimizer, but during execution by runtime as well.

-Spreading data across multiple table spaces. Can help the administration task backup/restore.

-Most of the scenarios handled by Union all view (UAV) today should be able to be replaced with using range partitioned table. The advantage is that all the complicated parts of using UAV are taken care of for you internally. From an "Application Development" perspective, there is no difference between a partitioned and a nonpartitioned table. The application need not be modified to access a partitioned table if that application was initially written on a non partitioned tables.



This slide shows the grand unification of all partitioning capabilities provided by DB2. All partitioning techniques can be used independently, or mixed and matched together.

When distributing by hash (DPF), this is the first determined so that the row can be sent to or retrieved from the correct database partition.

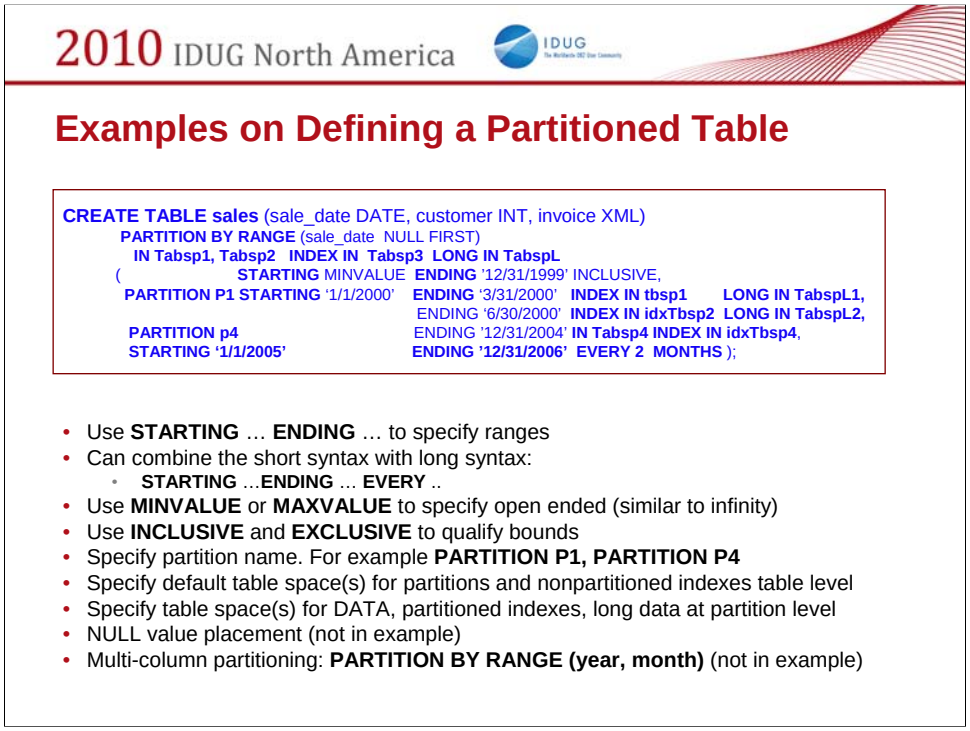
When partitioning by range (table partitioning), the database partition is then determined so that the correct range of the table is accessed.

When organizing by dimensions (MDC), this is the last concept applied in order to access the correct cell (composite dimension value).

Use, Benefits, and Considerations of Partitioning

- **Large tables (warehousing or other)**
  - Data rolled-in by range or lower granularity (use ATTACH or ADD)
  - Data rolled-out periodically using DETACH
  - Benefit to query predicates, partition elimination
  - Small or non-existent maintenance windows
- **Easy management of partitions**
  - ADD, ATTACH, DETACH for roll-in and roll-out
  - SET INTEGRITY online to maintain indexes & validate range after ATTACH
  - Table space usage can work well with partitioning strategy
    - BACKUP, RESTORE, and REORG utility strategies around partitions
    - Different storage media (e.g. disk vs. SSD) for different partitions
- **Business intelligence style queries**
  - Queries to roll up data by e.g. date, region, product, category
  - Queries are complex and/or long running
  - Table partition elimination for many queries

Set the expectations for how the feature is intended to be used.



You can see many aspects of the syntax which enable you to specify things differently in different business application scenarios accordingly.

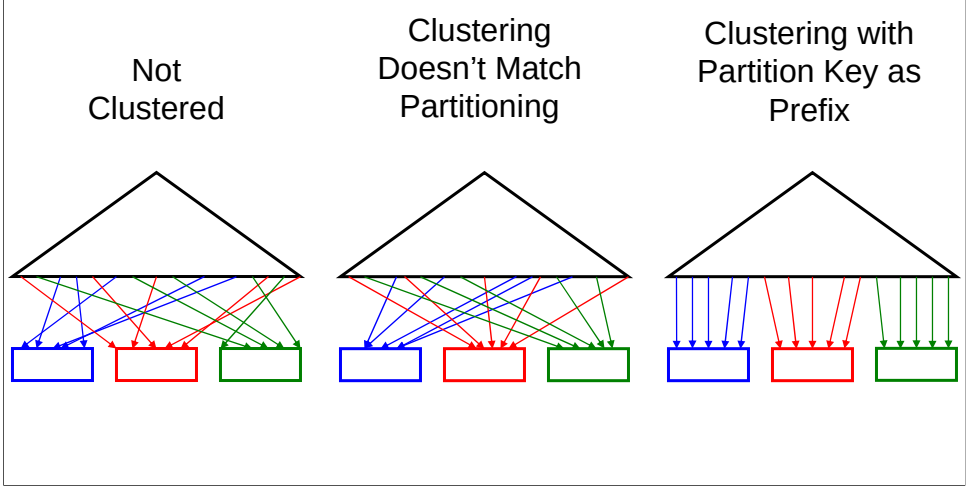
SYSCAT.DATAPARTITIONS Catalog View

- Catalog view contains information on each data partition
  - Partition names, table spaces, ranges, etc.
  - Only 1 entry per data partition, even if multiple database partitions
- *Some* statistics for each data partition (physical stats)
  - SYSCAT.TABLES has aggregate table stats used by optimizer
- Nonpartitoined tables **also** have 1 entry in this view

DESCRIBE Command (also ADMIN_CMD)

- **DESCRIBE DATA PARTITIONS FOR TABLE <schema.name> [SHOW DETAIL]**
  - Reports basic information about data partitions (ranges)
  - Acquires information from SYSCAT.DATAPARTITIONS
  - Output ordered by range order/sequence (low to high values)
  - SHOW DETAIL includes partition names, table spaces, status, etc
- **DESCRIBE TABLE <schema.name> SHOW DETAIL**
  - Enhanced to show table partitioning columns

Nonpartitioned Indexes - Clustered Indexes



Optimal scans are achieved by making the clustering match the partitioning key (right hand picture)

Sometimes, there is good reason to cluster on some other key

This can still work well. Even when clustering doesn't match the partitioning key (middle picture) there is still plenty of benefit

- Within each partition, rows are in key order

- What the scan will see, is each fetch will come from a set of n pages in the buffer pool where n is number of partitions

Contrast with a scan of clustered index in a non-partitioned table, where all reads come from same page until it is consumed

- As long as there is room for n pages to stay in the buffer pool, this works pretty well

The agenda of the session.

First up is a description of what things DB2 locks, how locks are physically represented, and which configuration parameters are used to manage lock memory.

ADD New Partitions to a Partitioned Table

```
ALTER TABLE sales ADD PARTITION
     starting ('2010-01-01') ending ('2010-01-31')
```

- Defines, creates, and adds a new *empty* partition to the partitioned table
  - Must specify range constraint
  - Optionally specify partition name and table spaces (vs. taking defaults)
  - Entry inserted into SYSCAT.DATAPARTITIONS
- A new index partition is created with empty indexes when any partitioned indexes exist on the partitioned table
  - Entry for the each partitioned index inserted into SYSCAT.SYSINDEXPARTITIONS
- Catalogs rows and partitioned table are locked (COMMIT for availability)
- No SET INTEGRITY is required
  - Data belonging to new **range** can be inserted, imported, or loaded into the ***table***

You can add empty partitions to a partitioned table using the ADD PARTITION option of the ALTER TABLE statement. Under the covers, a new table is created and logically made part of the partitioned table, updating the catalogs appropriately.

The most exciting feature within table partition is enhanced roll-in and roll-out. We have created two new operations to accomplish this.

ALTER TABLE ... ATTACH takes an existing table and incorporates it into a partitioned table as a new range. Authority required for ATTACH:

- For the target table:
  - ALTER + INSERT
- For the source table, one of the following:
  - SELECT (table) and DROPIN (schema), or …
  - CONTROL privilege, or …
  - SYSADM or DBADM authority

ALTER TABLE ... DETACH is the inverse operation. It takes one range of a partitioned table and splits it off as a stand alone table. Authority required for DETACH:

- For the source table:
  - ALTER + SELECT + DELETE
- For the target table, one of the following:
  - SYSADM or DBADM or …
  - CREATETAB (database) and USE (tablespace) as well as one of
  - IMPLICIT_SCHEMA
  - CREATEIN

The key point about these operations is that there is no data movement. Internally, they are mostly just manipulating entries in the system catalogs. This means that the actual ATTACH or DETACH operations are very fast - on the order of a few seconds at most.

Use SET INTEGRITY to Complete the Roll-in

- SET INTEGRITY is required after ATTACHing partitions to the partitioned table. SET INTEGRITY performs the following operations
  - Index maintenance
    - Incrementally maintain nonpartitioned indexes
    - Build any missing partitioned indexes
  - Checking of range and other constraints
  - MQT maintenance
  - Generated column maintenance
- Table is online through out process of running SET INTEGRITY
  - Use ALLOW WRITE ACCESS. Default is the original offline behavior
- New data becomes available on COMMIT of SET INTEGRITY
- Provide an exception table for SET INTEGRITY
  - Without an exception table, *any violation will fail the entire operation*

Without an exception table, any violation will fail the entire operation.
Recommendation: provide an exception table for SET INTEGRITY

Example:

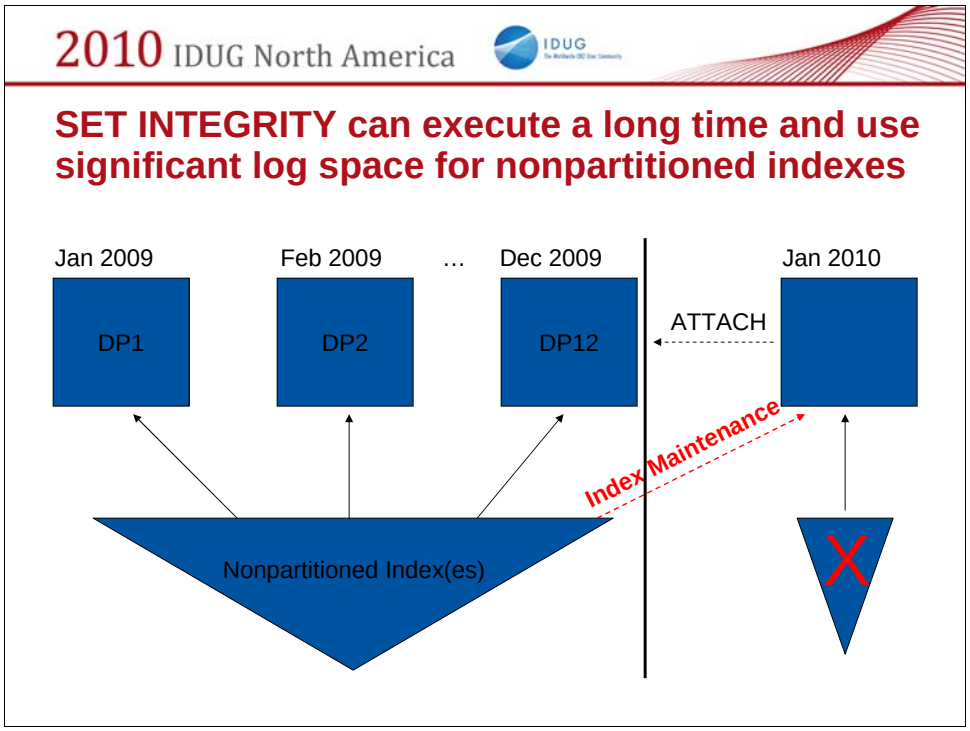
```
SET INTEGRITY FOR sales ALLOW WRITE ACCESS,
sales_by_region ALLOW WRITE ACCESS
IMMEDIATE CHECKED INCREMENTAL
FOR EXCEPTION IN sales USE sales_ex;
```

New behaviors specific to XML data with use of Exception Tables

When an exception table is specified, it stores rows that violate constraints in the tables being checked. The table is taken out of set integrity pending state even if errors are detected. A warning message to indicate that one or more rows have been moved to the exception table is returned (SQLSTATE 01603).

The rows are moved out of the target table and into the exception table via DELETE and INSERT

A new constraint violation type code 'X' in the message column of the exception table is introduced for SET INTEGRITY to denote an XML values index violation.



DP1 = Data Partition 1, …

The agenda of the session.

First up is a description of what things DB2 locks, how locks are physically represented, and which configuration parameters are used to manage lock memory.

The most exciting feature within table partition is enhanced roll-in and roll-out. We have created two new operations to accomplish this.

ALTER TABLE ... ATTACH takes an existing table and incorporates it into a partitioned table as a new range. Authority required for ATTACH:

- For the target table:
  - ALTER + INSERT
- For the source table, one of the following:
  - SELECT (table) and DROPIN (schema), or …
  - CONTROL privilege, or …
  - SYSADM or DBADM authority

ALTER TABLE ... DETACH is the inverse operation. It takes one range of a partitioned table and splits it off as a stand alone table. Authority required for DETACH:

- For the source table:
  - ALTER + SELECT + DELETE
- For the target table, one of the following:
  - SYSADM or DBADM or …
  - CREATETAB (database) and USE (tablespace) as well as one of
  - IMPLICIT_SCHEMA
  - CREATEIN

The key point about these operations is that there is no data movement. Internally, they are mostly just manipulating entries in the system catalogs. This means that the actual ATTACH or DETACH operations are very fast - on the order of a few seconds at most.

DETACH – Asychronous Index Cleanup

- AIC is a feature to cleanup indexes of detached content
  - Low priority background process
  - Reclaims index space (keys corresponding to detached partition)
  - Automatically started when DETACH is committed
    - or after refresh of dependent MQTs
  - Pauses when required
    - Lock conflict with user activity
    - Deactivate database
- AIC thread per partition, per (NP) index, per DB partition
- Monitored through LIST UTILITIES [SHOW DETAIL]
- No ADD/ATTACH partition with *same name* until AIC has completed cleaning all indexes for a particular partition

The agenda of the session.

First up is a description of what things DB2 locks, how locks are physically represented, and which configuration parameters are used to manage lock memory.

Materialized Query Tables (MQTs)

- SET INTEGRITY after ATTACH does all maintenance:
  - Index maintenance
  - Checking of range and other constraints
  - MQT maintenance
  - Generated column maintenance
- "Refresh immediate" MQTs go offline after DETACH
  - Use SET INTEGRITY on each MQT to refresh and bring online
  - Target table of DETACH is not usable until MQTs are dealt with
    - SYSCAT.DATAPARTITIONS shows 'D' in STATUS field
    - SYSCAT.TABLES shows 'L' for table type
  - Target can be made available via SET INTEGRITY option
    SET INTEGRITY … FULL ACCESS
    Note: this forces MQTs to be full processed

Table Partitioned MQTs

- MQTs can also be range partitioned
  - Same advantages as for base tables
    - Easy roll-in/roll-out
    - Query performance via partition elimination
    - Flexible space management
  - Correct ranges must exist, or you can get SQL0327N "out of bounds"
- DETACH and ADD are supported
- ATTACH is not supported on partitioned MQTs
  - ATTACH on the MQT is not needed if refreshed via SET INTEGRITY
  - For cases of user maintained MQTs, there is a workaround:
    - Convert MQT to plain table
    - Roll-in to base and MQT
    - Use IMMEDIATE UNCHECKED to restore MQT

Referential Integrity (RI)

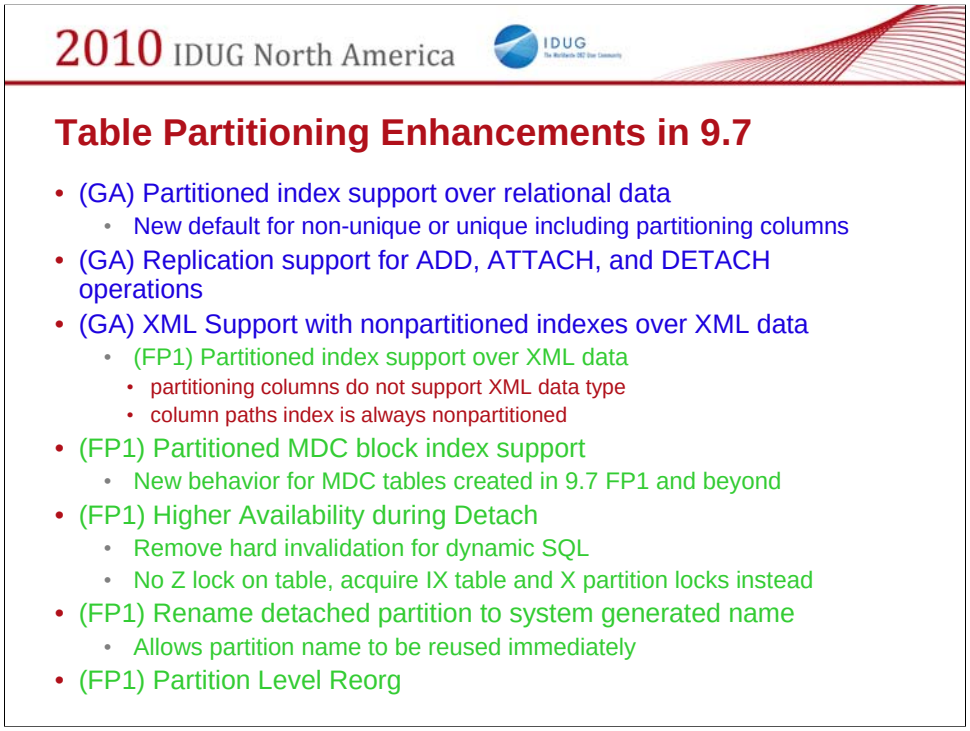
- ATTACH to child will validate parent data during SET INTEGRITY
- DETACH from the parent is not allowed
  - Would expect that parent table is not commonly partitioned
  - There is a workaround:

```
ALTER TABLE c ALTER FOREIGN KEY fk NOT ENFORCED;
ALTER TABLE p DETACH PARTITION p0 INTO TABLE pdet;
ALTER TABLE c ALTER FOREIGN KEY fk ENFORCED;
SET INTEGRITY FOR c ALL IMMEDIATE UNCHECKED;
COMMIT WORK;
```

  - Do all this in a single transaction to lock out concurrent updates
  - User responsible to guarantee that RI is NOT broken in the process!

The agenda of the session.

First up is a description of what things DB2 locks, how locks are physically represented, and which configuration parameters are used to manage lock memory.



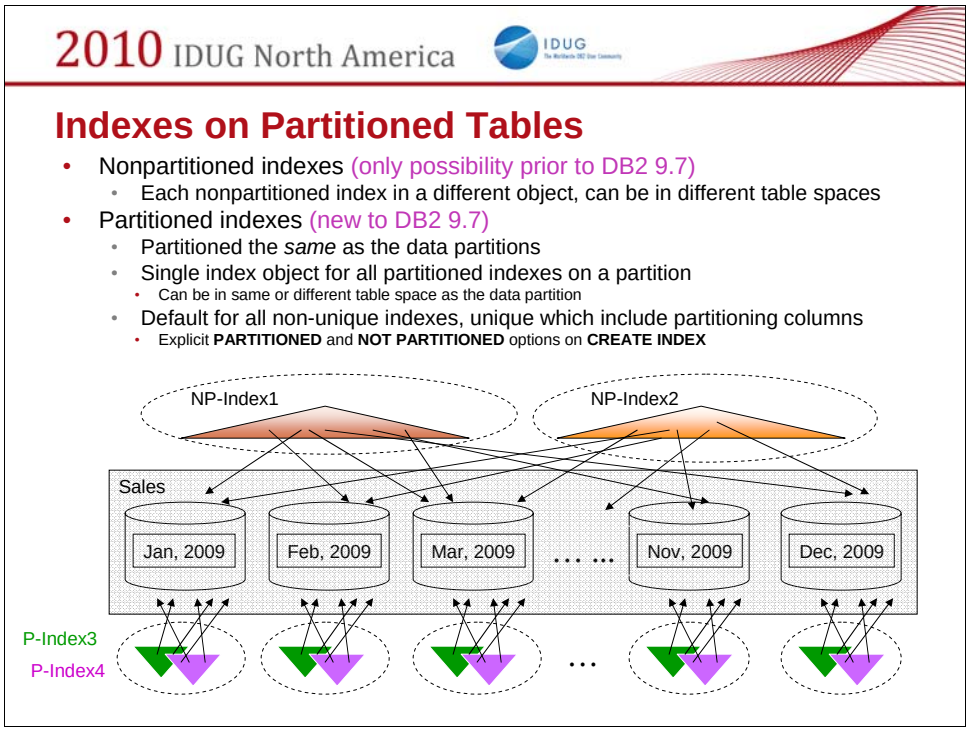
The table partitioning feature in DB2 9 provides a number of advantages, particularly to the data warehouse (DW) applications. DW systems will benefit from easier roll-in/roll-out of data and better query execution performance. DW systems currently using union all views are particularly well suited to make use of table partitioning.

The table partitioning enhancements in DB2 9.7 include partitioned indexes, higher availability during DETACH, partition level reorganization support on both data and indexes, XML columns, etc.

Some of these enhancements are provided in DB2 9.7 GA, while most have been delivered as part of DB2 9.7 FP1.

The agenda of the session.

First up is a description of what things DB2 locks, how locks are physically represented, and which configuration parameters are used to manage lock memory.



UNIQUE IDNEX:

A unique index (and therefore unique or primary key constraints being enforced using system generated unique indexes) cannot be partitioned unless the index key columns are a superset of the table partitioning key columns. That is, the columns specified for a unique key must include all the columns of the table partitioning key (SQLSTATE 42990).

DEFAULT:

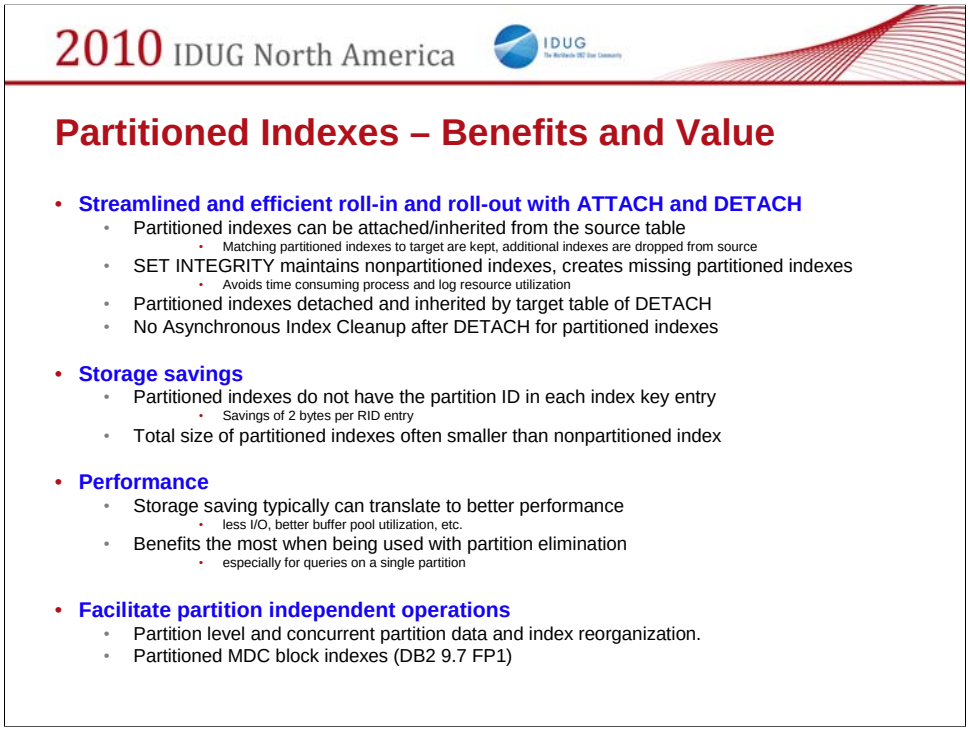
When the table is partitioned and CREATE INDEX does not specify either PARTITIONED nor NOT PARTITIONED keywords CREATE INDEX will create a partitioned index by default unless:

1. an unique index is being created *and* the index key does not include all the table partitioning key columns

2. a spatial index is being created

Index over XML data:

User created partitioned indexes over XML data only supported in 9.7 FP1.



Streamlined roll-in/roll-out with ATTACH/DETACH

Partitioned index can be attached along with the data

Partitioned index on the source can be kept and logically converted to the partitioned index on the newly ATTACHed partition.

Set Integrity doesn't need to maintain partitioned indexes

Strictly speaking, this is only true when the source table has all the indexes pre-created before ATTACH to match all the partitioned index on the target *(Best Practices).*

Partitioned index will be detached along with the data

In another word, they can be inherited by the target table of DETACH

No AIC is necessary after DETACH for partitioned indexes

The major drawbacks for partitioning indexes is it losts order for some queries when the partitioning column is not the leading column of index keys.

The indexes can not help the order requests in this case and extra sorts are required.

SYSCAT.INDEXPARTITIONS Catalog View

- Catalog view contains information on each index partition
  - Table and index names, data partition ID, table space, etc.
  - Only 1 entry per data partition, even if multiple database partitions
- *Most* statistics for each index partition
  - SYSCAT.INDEXES has aggregate index stats used by optimizer
- Nonpartitoined indexes and indexes on nonpartitioned tables do ***not*** have an entry in this view!

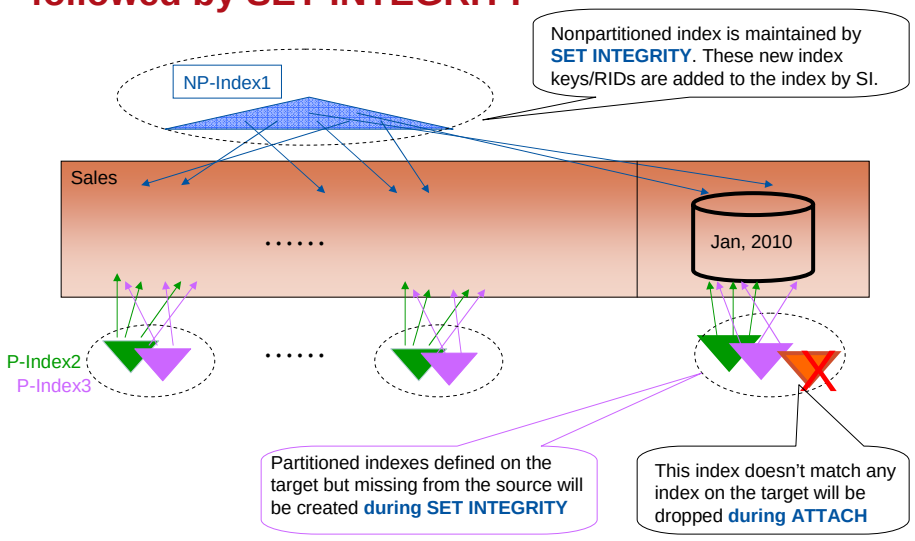
DESCRIBE Command (also ADMIN_CMD)

- **DESCRIBE INDEXES FOR TABLE** command is extended to report if an index is partitioned or not with a new column “Index Partitioning” which can have values:
  - 'N‘ = This is a nonpartitioned index on a partitioned table
  - 'P‘ = This is a partitioned index on a partitioned table.
  - ‘ ’ = Index is not on a partitioned table
- Sample output: DESCRIBE INDEXES FOR TABLE *myDPartT*

```
Index            Index           Unique    Number of    Index
schema           name            rule      columns      Partitioning
---------------  --------------- --------  -----------  ------------
NEWTON           IDXNDP          D                   1  N
NEWTON           IDXDP           D                   1  P
```

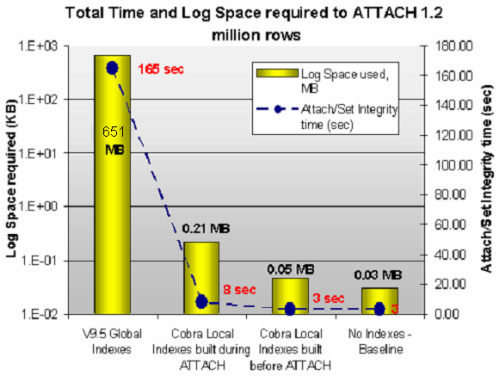
- **DESCRIBE DATA PARTITIONS FOR TABLE** now includes the new column **IndexTblSpId**; the table space ID associated and used for the index partition.
- **ADMIN_GET_INDEX_INFO()** updated to report if an index is partitioned or not.

ALTER TABLE … ATTACH PARTITION … followed by SET INTEGRITY



Roll-in (ATTACH): Savings at a glance

- Attached 1.2 millions rows, table partitioned by week
- Partition maintenance
  - **20x speedup** compared to 9.5 nonpartitioned index because of reduced index maintenance
  - **3000x less log space** used than with nonpartitioned index
- **BEST PRACTICE**
  - Use partitioned indexes (unless unique without partitioning columns)
  - Prepare source table with indexes matching partitioned index of target partitioned table



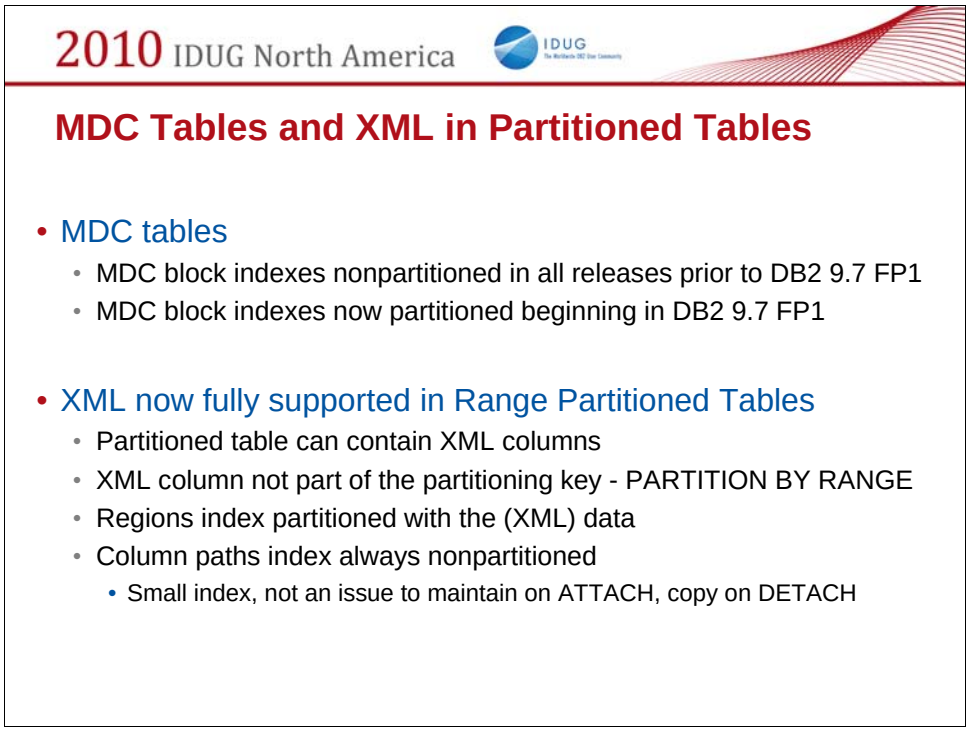
Online CREATE INDEX for Partitioned Indexes

- Partitioned indexes are created 1 partition at a time, online for read/write for each, then the partition becomes read only as the partitioned index is completed, and remains read only until the transaction completes.

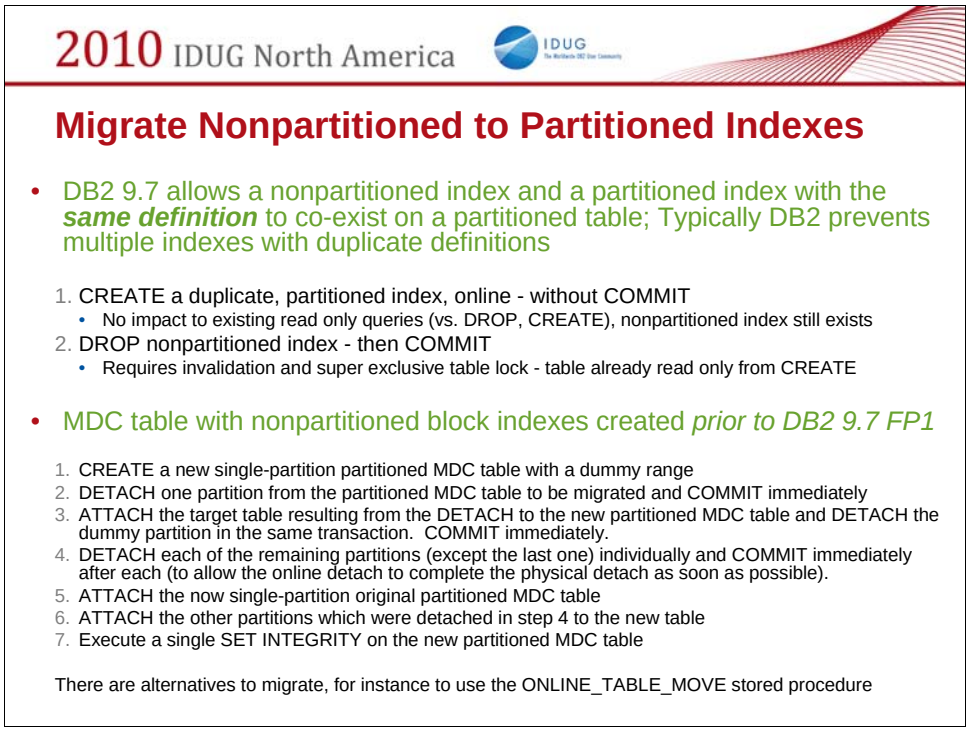
1. All data partitions available for read and write
2. For each data partition, starting at the lowest sequence number (SEQNO)
    3. Build index partition (data partition continues to be available for read/write during this step)
    4. Quiesce data partition to read-only (S partition lock)
    5. Perform catch-up for changes made to the data partition during index partition creation
6. Repeat steps 3-5 for each data partition
7. After the last data partition is built and the transaction is committed, all data partitions are available again for full read and write

- With the common usage of table partitioning, partition a table on date, most update activities occurring on the most recent partition, this approach to online index creation for partitioned indexes is simple and practical.

To be after slide 7 – benefits of partitioned indexes



XML column is not allowed prior cobra. Including xml column in an MDC table definition resulting and error. We are not trying to cluster the XML data itself or make it XML column as part of the clustering key.

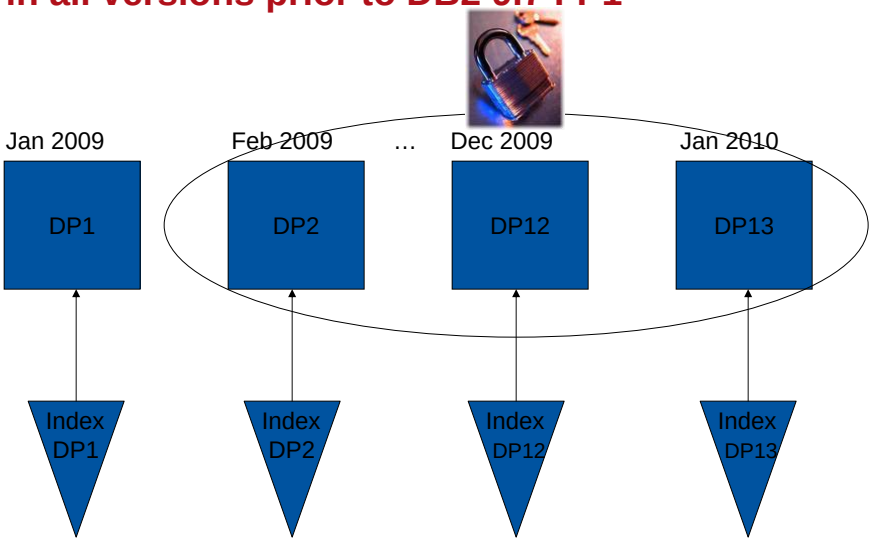


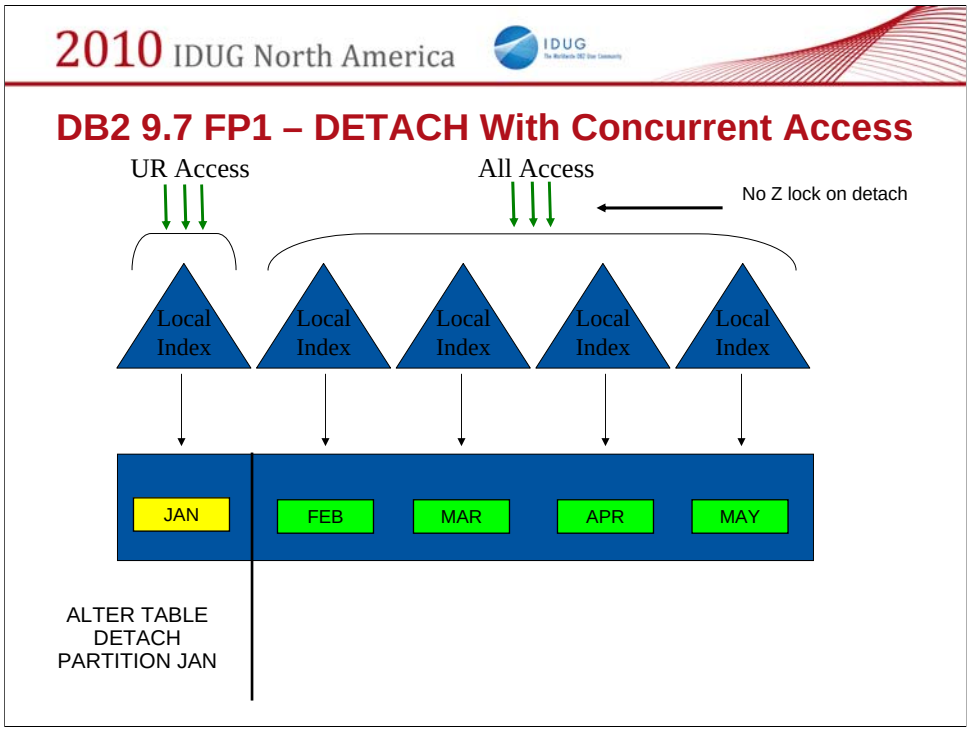

```
connect to mydb;

-- Assume I have a partitioned MDC from v97 GA or before;
-- the MDC block indexes are global;
-- one can simulate this by the internal variable to create global MDC block indexes
-- db2set
DB2_INDEX_DEFAULT=MDC_BLOCK_INDEXES_NONPARTITIONED;

create table myOldT (i1 int, i2 int, i3 int) partition by range (i1) (starting 1 ending 10 every 2) organize by dimensions(i2, i3);
insert into  myOldT values (1, 2, 1),(3, 2, 2),(5, 2,3),(7,2,4),(9,2,5);
commit;

-- Verify from the cat table,  the mdc block indexes are global index
db2 => select * from sysindexes;
select * from sysindexes

TBSCHEMA   TABNAME    IDXSCHEMA          INDNAME    INDEXTYPE TBSPACEID   IDXOBJID    IID
---------- ---------- -------------------- ---------- --------- ----------- ----------- ------
LIPING     MYOLDT     SYSIBM             SQL0909120 BLOK            4         9     1
LIPING     MYOLDT     SYSIBM             SQL0909120 DIM             4        10     2
LIPING     MYOLDT     SYSIBM             SQL0909120 DIM             4        11     3

  3 record(s) selected.


connect reset;

-- Now turn the default block index internal regirstry var off, so MDC block indexes will be partitioned
db2set DB2_INDEX_DEFAULT=;
db2stop;
db2start;

connect to mydb;

-- We want to convert it to a new partitioned MDC table using local index
```

The agenda of the session.

First up is a description of what things DB2 locks, how locks are physically represented, and which configuration parameters are used to manage lock memory.

DETACH waits for and blocks all table access in all versions prior to DB2 9.7 FP1





Detaching a partition turns that partition into it's own table. To attach a new partition you create a table with the same definition, load it with data and then ATTACH it to the partitioned table.

The detach and/or attach commands act virtually instantaneously. For nonpartitioned indexes (none shown in this example) on the table, index keys are physically removed asynchronously in the background. However, as part of the DETACH statement itself these index keys are logically removed (that is, an index scan will not see these keys). Similarly an attach occurs almost immediately and then the index keys for nonpartitioned indexes are added with the set integrity command. The SET INTEGRITY statement allows full read/write access to the previously existing partitions.

New Two-phase DETACH in DB2 9.7 FP1

Phase 1

- Logically delink the partition from the table. This phase happens as part of the ALTER TABLE … DETACH statement
  - Soft invalidation of all dynamic SQL
  - Hard invalidation of all static SQL
  - Existing UR isolation access is allowed on detached partition
  - IX table lock, X partition lock (all data in partition deleted from table)
  - Modify catalogs to remove partition from table definition
  - Rename detached partition to system name to allow name reuse
  - New SQL compilation allowed following COMMIT of DETACH

Phase 2

- After COMMIT of DETACH, asynchronously physically delink the partition from the table, using Asynchronous Background Processing (ABP) infrastructure. This is referred as Asynchronous Partition Detach (APD).
  - Hard invalidation of any previously soft invalidation access plans
  - Finalize detached partition into the standalone table
  - Asynchronous Index Cleanup (AIC) on any nonpartitioned indexes

The agenda of the session.

First up is a description of what things DB2 locks, how locks are physically represented, and which configuration parameters are used to manage lock memory.

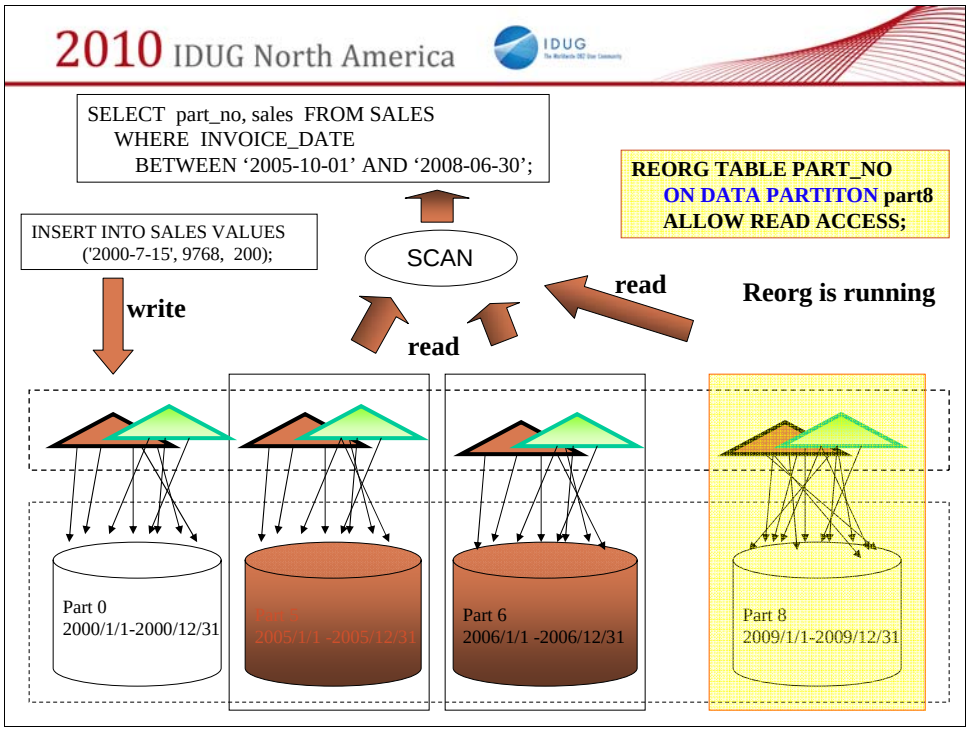
Reorg Table and Indexes

- REORG TABLE for a partitioned table is always offline
- REORG INDEXES ALL on a partitioned table is always offline
- REORG INDEX reorganizes a nonpartitioned index, supporting all access modes

- **Partition level reorg table and indexes are available in 9.7 FP1**
  - REORG TABLE <t-name> **ON DATA PARTITION** <part-name>
    - **ALLOW NO/READ ACCESS** applies to part-name, not the entire partitioned table
  - **Without** nonpartitioned indexes (except xml path), the ALLOW READ ACCESS mode is the default behavior with full read/write access to all other partitions
  - **With** nonpartitioned indexes (except xml path), ALLOW NO ACCESS is the default and only supported mode.

  - REORG INDEXES ALL FOR TABLE <t-name> **ON DATA PARTITION** <part-name>
    - **ALLOW NO/READ/WRITE ACCESS** applies to part-name, not the entire partitioned table
    - **ALLOW WRITE ACCESS** is *not* supported for MDC tables (w/ mdc block indexes).

- ***Concurrent partition reorg (TABLE and INDEXES ALL) supported when there are no nonpartitioned indexes (excluding XML column paths index) and ALLOW NO ACCESS is specified***


SELECT part_no, sales FROM SALES
WHERE INVOICE_DATE
BETWEEN '2005-10-01' AND '2008-06-30';
REORG TABLE PART_NO
ON DATA PARTITON part8
ALLOW READ ACCESS;
INSERT INTO SALES VALUES
('2000-7-15', 9768, 200);
SCAN
write
read
read
Reorg is running
Part 0
2000/1/1-2000/12/31
Part 5
2005/1/1 -2005/12/31
Part 6
2006/1/1 -2006/12/31
Part 8
2009/1/1-2009/12/31

REORGCHK Command and Stored Procedures

- SYSCAT.[DATAPARTITIONS|INDEXPARTITIONS]
  - Contain partition specific physical statistics
- With new partition level REORG capability, individual partition reorg recommendations are now included
  - REORGCHK command
  - REORGCHK_TB_STATS stored procedure
  - REORGCHK_IX_STATS stored procedure

Potential Future Enhancements: What's on the Whiteboard at the Lab? What do you want or recommend we consider?

- Performance
- Function
- Flexibility
- Usability
- Availability
- …
- …
- …

Speaker Biography

Mike Winer is a Senior Technical Staff Member and Kernel Architect in DB2 for LUW. During his 17+ years experience in DB2, Mike has worked on a number of key DB2 components such as locking, logging, data and index management, recovery, and utilities. His experience in performance, development and database architecture, together with the implementation of numerous key features and capabilities in DB2 have established him in his current role as one of the lead architects in the DB2 team.

Most recently Mike has been an architect for Deep Compression, Currently Committed, Partitioned Indexes, Inline LOB, Reads on HADR Standby, among many others.

Which MQTs Need to Have SET INTEGRITY?

After ATTACH or DETACH, you need to run SET INTEGRITY on all dependent REFRESH IMMEDIATE MQTs

```
WITH
DEP_CNT(TOTAL_DEP) AS (SELECT COUNT(*) FROM SYSCAT.TABDEP),
DEP_TAB(SCHEMA, NAME, TYPE, REFRESH, LEVEL) AS
  (SELECT TABLES.TABSCHEMA, TABLES.TABNAME, TABLES.TYPE,
          TABLES.REFRESH, 0
   FROM SYSCAT.TABLES TABLES
   WHERE TABLES.TABSCHEMA='<schema name>' AND TABLES.TABNAME='<table name>'
     UNION ALL
   SELECT TABDEP.TABSCHEMA, TABDEP.TABNAME, TABDEP.DTYPE,
          TABLES.REFRESH, DEP_TAB.LEVEL + 1
   FROM SYSCAT.TABDEP TABDEP, DEP_TAB, SYSCAT.TABLES TABLES
   WHERE TABDEP.DTYPE IN ('S', 'V', 'W', 'T') AND
         TABDEP.BSCHEMA = DEP_TAB.SCHEMA AND
         TABDEP.BNAME = DEP_TAB.NAME AND
         TABLES.TABSCHEMA = TABDEP.TABSCHEMA AND
         TABLES.TABNAME = TABDEP.TABNAME AND
         DEP_TAB.LEVEL < (SELECT DEP_CNT.TOTAL_DEP FROM DEP_CNT))
SELECT DISTINCT * FROM
(SELECT DEP_TAB.SCHEMA, DEP_TAB.NAME
 FROM DEP_TAB
 WHERE DEP_TAB.TYPE = 'S' AND DEP_TAB.REFRESH = 'I'
 ) X;
```