Partitioning: Tips and Tricks

Arup Nanda

Longtime Oracle DBA

Agenda

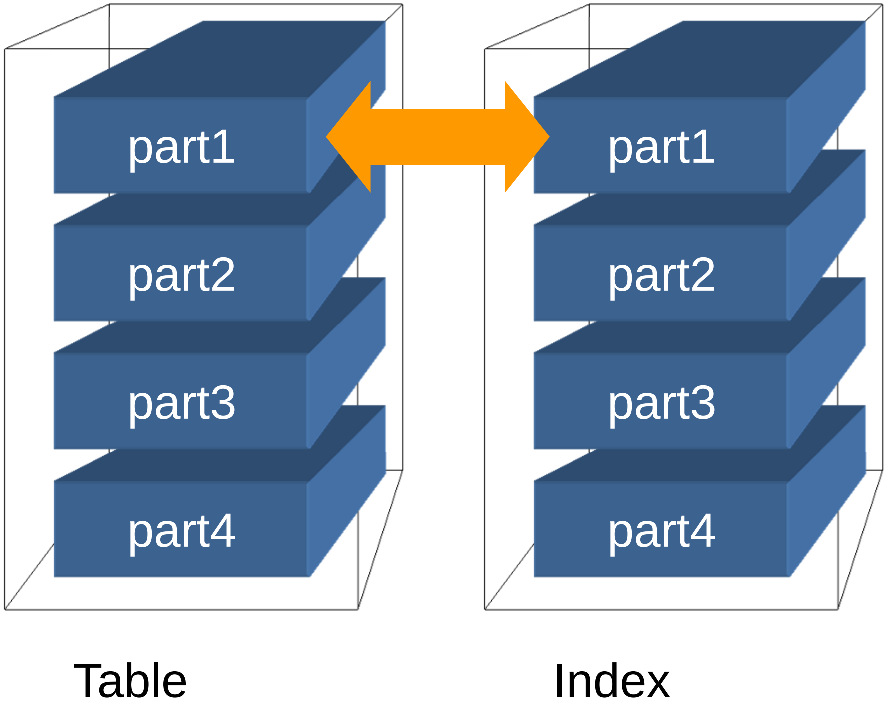
- Partitioning primer
- Choosing a partition strategy
- Choosing a partition key
- Solutions to common problems using partitioning
- Potential issues to watch out for
- Creative solutions in partitioning

Primer

Local Indexes

- Index is partitioned exactly as the table
- Index entries of each part are found in the corresponding partition in index only
- When table partition is dropped, so is the index partition
- Example

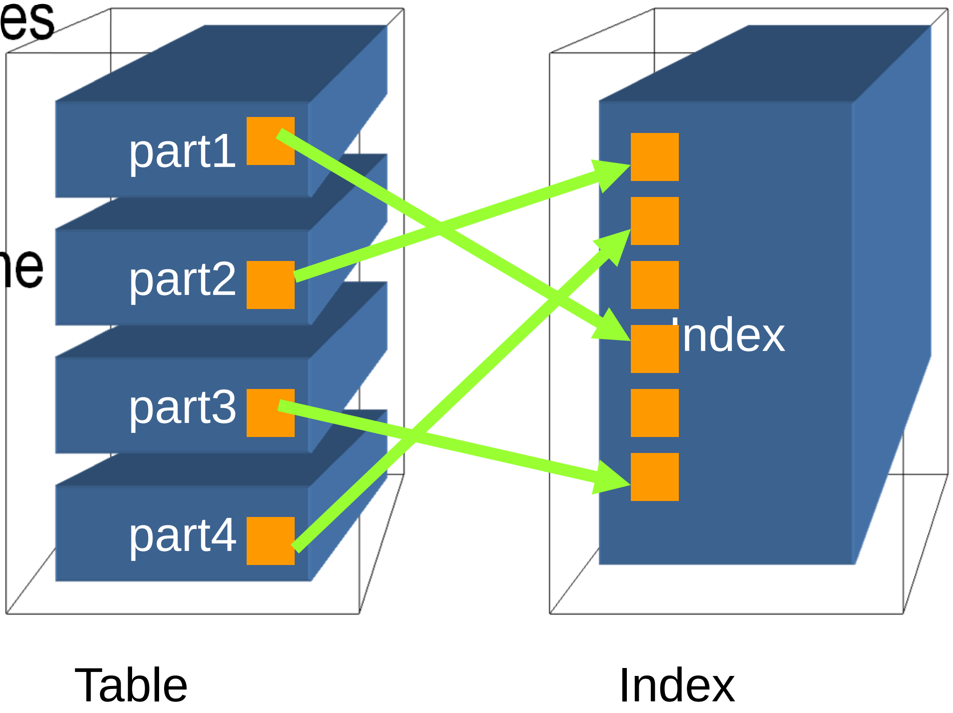
```
create index in_mytab
on mytab (col1)
local
```



Global Indexes

- Entries for all parts of the table are found all over the index.
- Usually used for unique indexes
- Index may be optionally partitioned
- When table part is dropped, the index needs to be rebuilt.
- Example

```
CREATE INDEX PK_MYTAB
ON MYTAB (COL2)
GLOBAL;
```



Global-vs-Local Index

- Whenever possible, use local index
- In Primary Key (or Unique) Indexes:
  - If part column is a part of the PK – local is possible and should be used
  - e.g. TXN table. PK – (TXN_DT, TXN_ID) and part key is (TXN_DT)
- If not, try to include the column in PKs
  - E.g. if TXN_ID was the PK of TXN, can you make it (TXN_DT, TXN_ID)?
- Ask some hard design questions
  - Do you really need a PK constraint in the DW?

Global indexes can be partitioned

- The global indexes can themselves be partitioned in any manner, different from the table partitioning scheme

```
create table mytab
(
   col1 number,
   col2 date,
   col3 varachar2,
   … and so on for other columns …
)
partition by range (col1)
(
  partition p1 values less than (101),
  partition p2 values less than (201),
  partition p2 values less than (301)
)
```

```
create index pk_mytab
on mytab (col2)
global
partition by hash
partitions 4;
```

Global index is hash partitioned while table is range partitioned, on different columns.

Different Range Partitioning

```
create table mytab
(…)
partition by range (col1)
(
partition p1 values less than (101),
partition p2 values less than (201),
partition p2 values less than (301)
)
```

```
create index IN1 on MYTAB (col4)
global
partition by range (col4)
(partition p1 values less than
(100),
partition p2 values less than
(maxvalue)
)
```

```
create index IN1 on MYTAB (col2)
global
partition by range (col4)
```

Will fail with ORA-14038: GLOBAL partitioned index must be prefixed

```
create index IN1 on MYTAB
(col4,col2) global
partition by range (col4)
```

Sub-Partitioning

- Range-Hash
  - Sales Date and Sales Trans ID
- Range-List
  - Sales Date and Product Code
- Range-range
  - 2 date columns
- List-range
  - Product code and then sales date
- List-list
  - Product code and geographic territory
- List-Hash
  - Product code and transaction id

Global Index Maintenance

- Global Indexes maintained with the partition operation

```
alter table mypart drop partition p1 update
indexes;
```

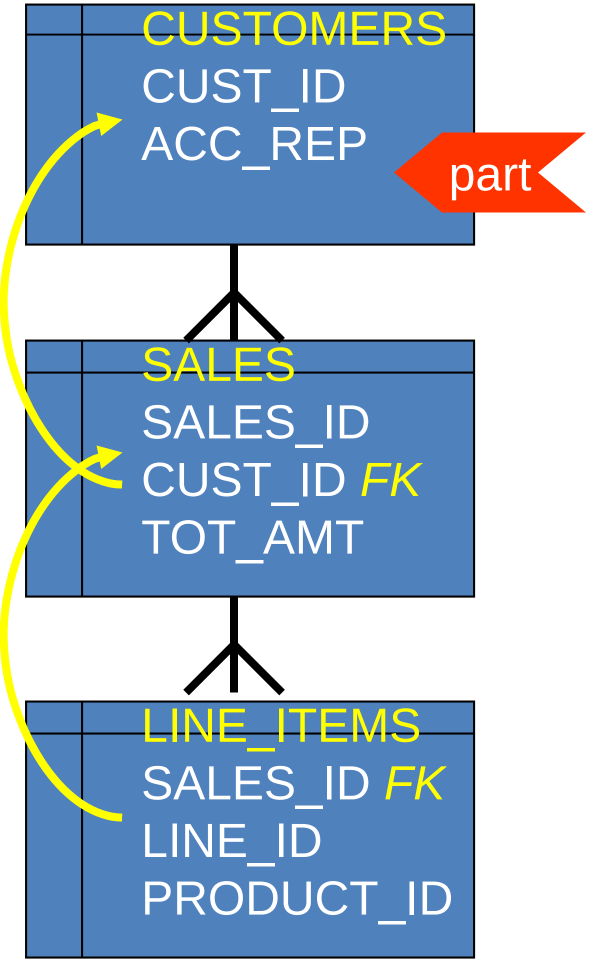
- Or, only global indexes:

```
alter table mypart drop partition p1 update
global indexes;
```

12c
NEW!
11
g
php
SQL
<XML>

Referential Partitioning

- You want to partition CUSTOMERS on ACC_REP column
- The column is not present on child tables
- Earlier option: add the column to all tables and update it
  - Difficult and error-prone
- 11g has referential partitioning

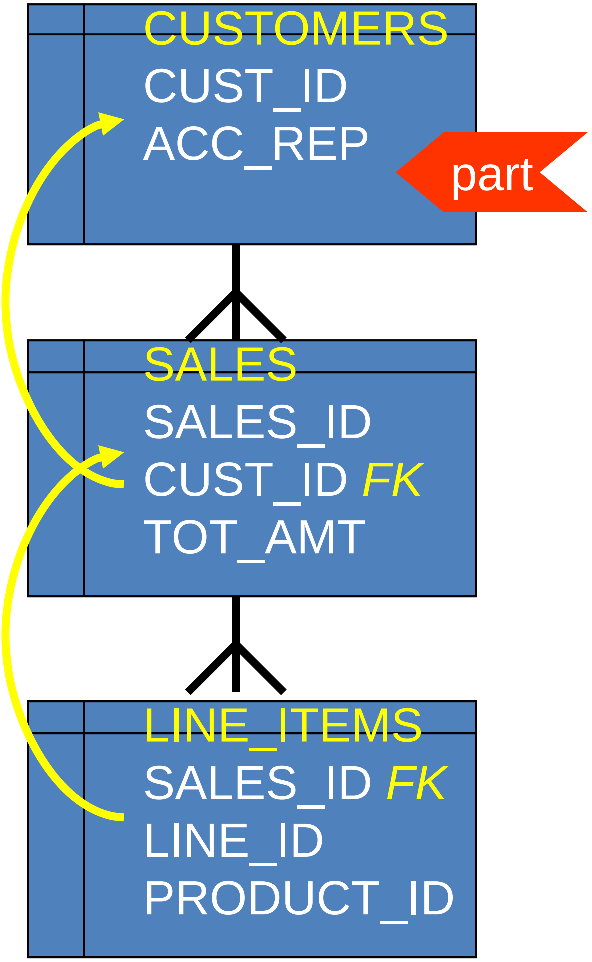


Referential Partitioning

Partition CUSTOMERS as usual

```
create table SALES (
  SALES_ID number not null,
  CUST_ID  number not null,
  TOT_AMT  number
  constraint fk_sales_01
    foreign key (cust_id)
    references customers)
partition by reference
   (fk_sales_01);
```

Partitions of SALES are created with data from CUSTOMERS.



Addressing Ref Partitions

- USER_PART_TABLES view has info
  - – partitioning_type – "REFERENCE"
  - – ref_ptn_constraint_name – the FK name
- The partitions will also bear the same name as the parent

INTERVAL Partitioning

- SALES table partitioned on SALES_DT
  - Partitions defined until SEP 2008. Before Oct starts, you have to create the partition
  - If you don't create the part, the INSERT will fail on Oct 1st.
- To mitigate the risk, you created the PMAX partition. *Undesirable*
- When you finally add the OCT08 partition, you will need to split the PMAX – *highly undesirable*

Interval Partitions

```
create table SALES ( sales_id number,
  sales_dt date )
partition by range (sales_dt)
interval (numtoyminterval(1,'MONTH'))
   store in (TS1,TS2,TS3)
( partition SEP08 values less than (to_date('2008-
  10-01','yyyy-mm-dd'))
);
```

Specifies one partition per month

This is the first partition. The subsequent partition names are system generated

Creates a partition automatically when a new row comes in

Addressing Interval Partitions

- USER_PART_TABLES view:
  - – partitioning_type – "INTERVAL"
- USER_TAB_PARTITIONS view:
  - – high_value shows the upper bound of partition
- To address a specific partition:

```
select * from SALES partition for
  (to_date('22-sep-2008','dd-mon-yyyy'));
```

Non-Interval Process

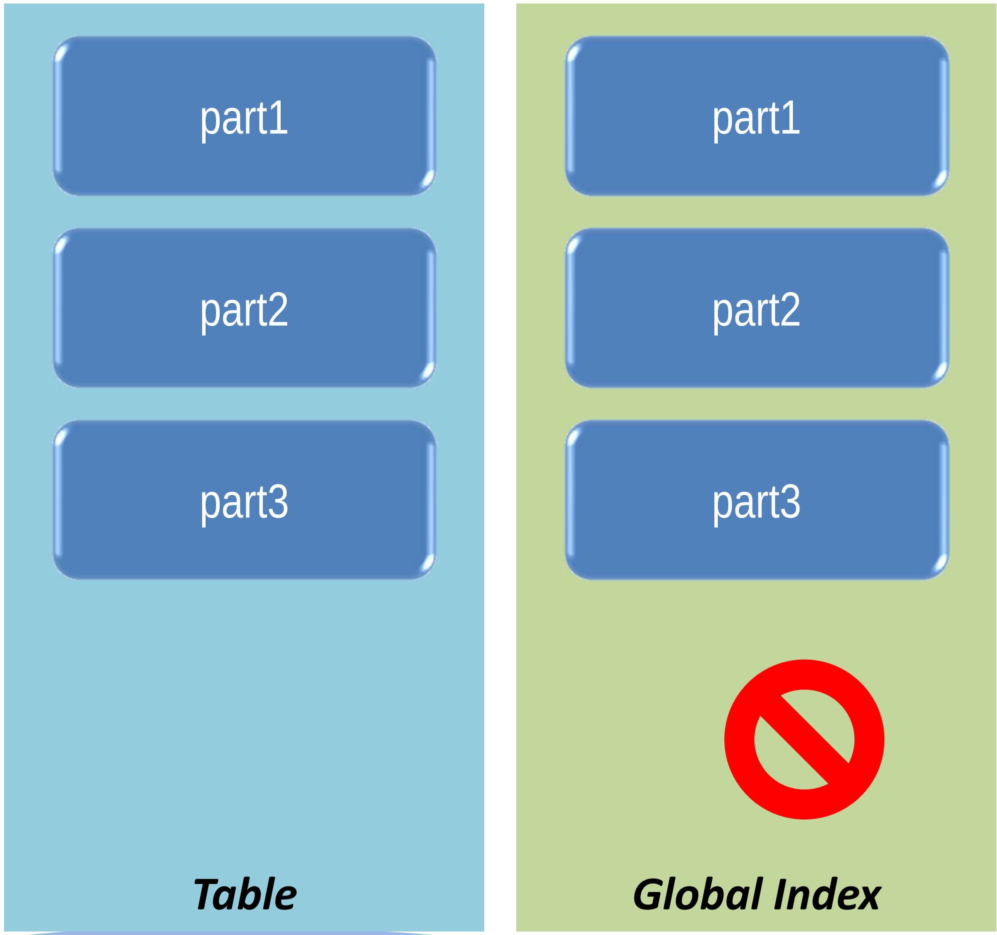
- To add partitions automatically:

  http://arup.blogspot.com/2010/11/tool-to-add-range-partitions.html

- To drop partitions automatically:

  http://arup.blogspot.com/2010/11/automatic-range-partition-dropping-tool.html

Asynch Global Index



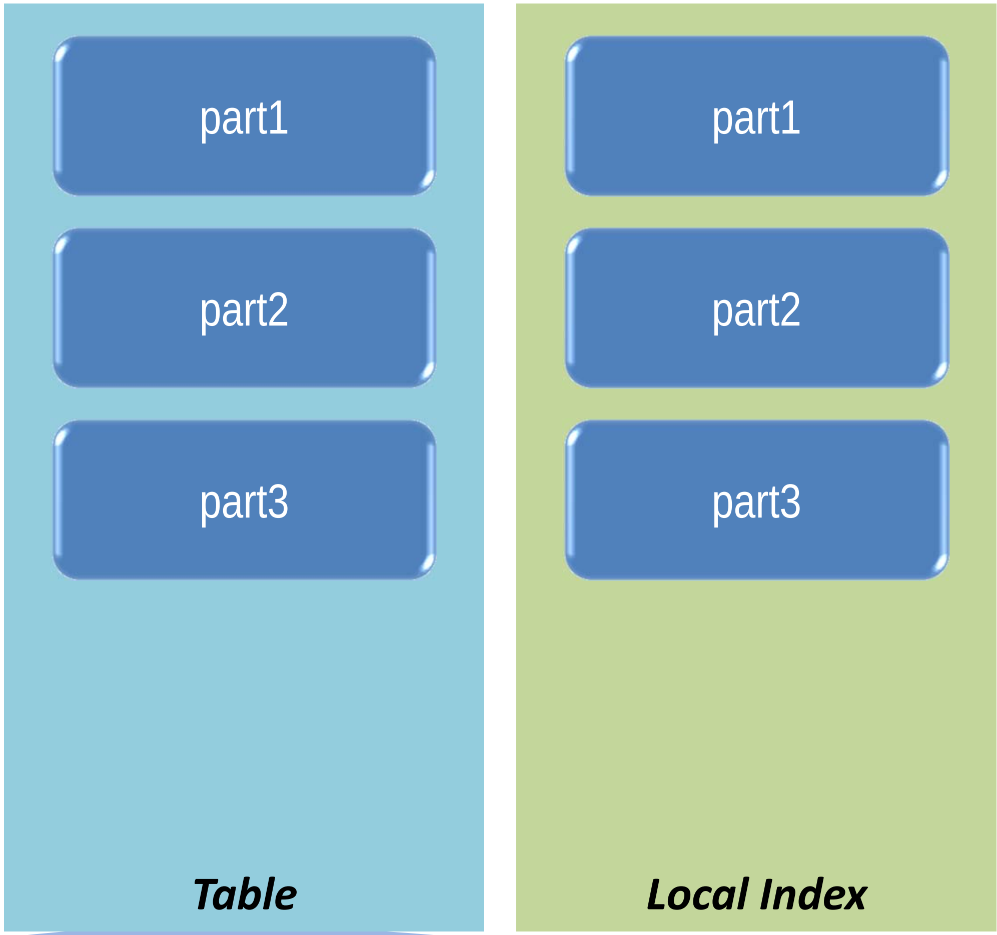
```
alter table drop t
partition part3
update global
indexes;
```

A scheduler job pmo_deferred_gidx _maint_job cleans up

Column ORPHANED_ENTRIES in USER_INDEXES view

giptab_test1.sql

Partial Index



```
SQL> alter table ptab1
modify partition p1
indexing on;

SQL> alter table ptab1
modify partition p2
indexing off;

SQL> create index
in_g2_ptab1 on ptab1
(c1) global indexing
partial;
```

partindex_test1.sql

Watchout!

Date Partition-keys

- Clear definition helps
- This will not choose the partition at compile time

```
where sales_date between '1-jan-09' and '31-jan-09';
```

- This will:

```
where sales_date between
TO_DATE(' 2009-01-01 00:00:00', 'SYYYY-MM-DD
  HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
and
TO_DATE(' 2009-01-31 00:00:00', 'SYYYY-MM-DD
  HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
/
```

date_explain1.sql
date_explain2.sql
exp.sql

Partition-wise Joins

- Works for range partitioned tables
- Not for hash partitioned
- Works only for equality operators; not ranges

pwj_test1.sql
pwj_test2.sql

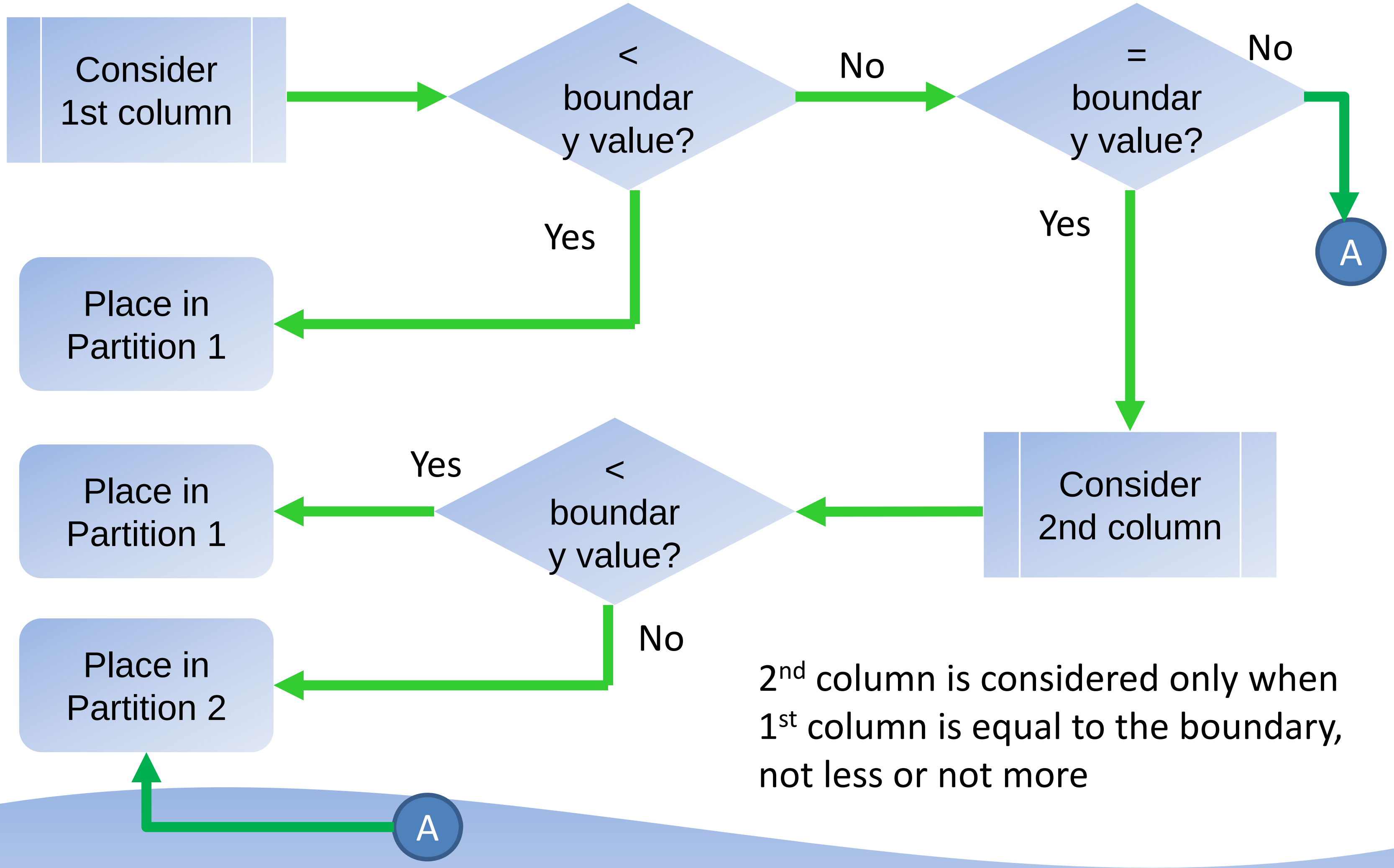
Multicolumn

```
create table mcptab1
(
   col1 number(10),
   col2 number(10)
)
partition by range (col1, col2)
(
   partition p1 values less than (101, 101),
   partition p2 values less than (201, 201)
)
```

```
      COL1       COL2
---------- ----------
       100        100
       100        101
       100        102
       101        100
       100        200
       100        201
       100        202
       101        101
       101        102
       102        100
       102        101
       102        102
       101        200
       101        201
       101        202
       102        200
       102        201
       102        202
```

mcpart_test1.sql

Multi-part key Determination



2nd column is considered only when 1st column is equal to the boundary, not less or not more

Arup Nanda

Subpart Stats Collection

- The normal method to collect stats

```
begin
    dbms_stats.gather_table_stats (
        ownname=> user, tabname=>'PTEST2');
end;
```

- Problem:
  - This populates the partition stats but not subpartition stats
  - To collect the subpartition stats, you must use granularity parameter. It has to be either ALL or SUBPARTITION

```
begin
    dbms_stats.gather_table_stats (
        ownname=> user, tabname=>'PTEST2‘, granularity=>’SUBPARTITION’);
end;
```

subpart_stats1.sql
subpart_stats2.sql

Partition stats collection

- The granularity parameter controls the scope for the stats
- Possible Values
    1. AUTO – determined by Oracle
    2. GLOBAL AND PARTITION – global stats and partition-level stats (subpartition level stats are not collected)
    3. SUBPARTITION – down to subpartition level
    4. GLOBAL – only global stats
    5. ALL – global, part and subpart level
    6. APPROX_GLOBAL AND PARTITION – new in 11g. Global stats are not calculated; but derived from partition stats

GRANULARITY	Table	Partition	Partition	Subpartition
	Global	Global	Statistics	Statistics
GLOBAL	YES	NO	NO	NO
PARTITION	NO	YES	YES	NO
DEFAULT	YES	YES	YES	NO
SUBPARTITION	NO	NO	YES	YES
ALL	YES	YES	YES	YES

Stats for a specific partition only

- To collect stats for a specific partition (or subpartition)
- Use the partname parameter

```
begin
    dbms_stats.gather_table_stats (
        ownname=> user, tabname=>'PTEST2‘, partname=>’SALES_Q1’,
);
end;
```

- In 11g, the global stats are automatically updated

Creative Solutions

Partition on Virtual Columns

- VC: not stored with the table
- Computed at runtime
- Can be indexed and partitioned

vcpart_test1.sql

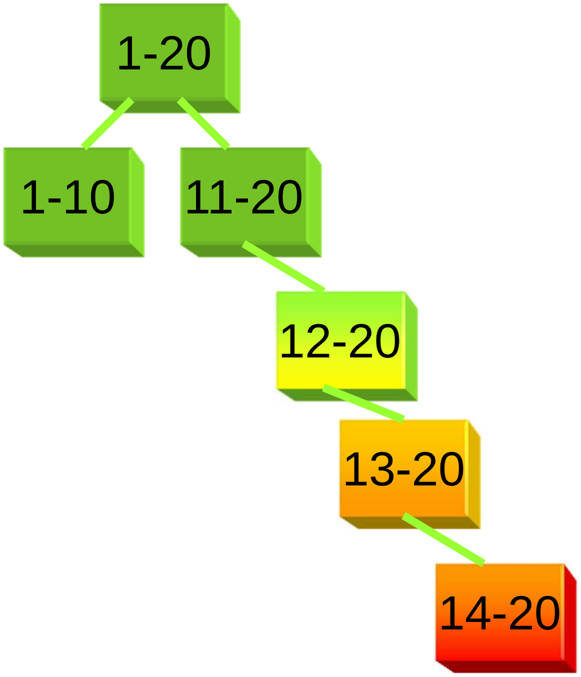
Partition on Invisible Columns

- Invisible columns are not visible
- Need not be entered
- Can be indexed and partitioned

Invcolpart_test1.sql

Index Blocks Too Hot to Handle

- Consider an index on TRANS_ID – a monotonically increasing number
- It may make a handful of leaf blocks experience severe contention
- This hot area shifts as the access patterns change
- Solution: Reverse Key Index?



Solution: Hash Partitioned Index

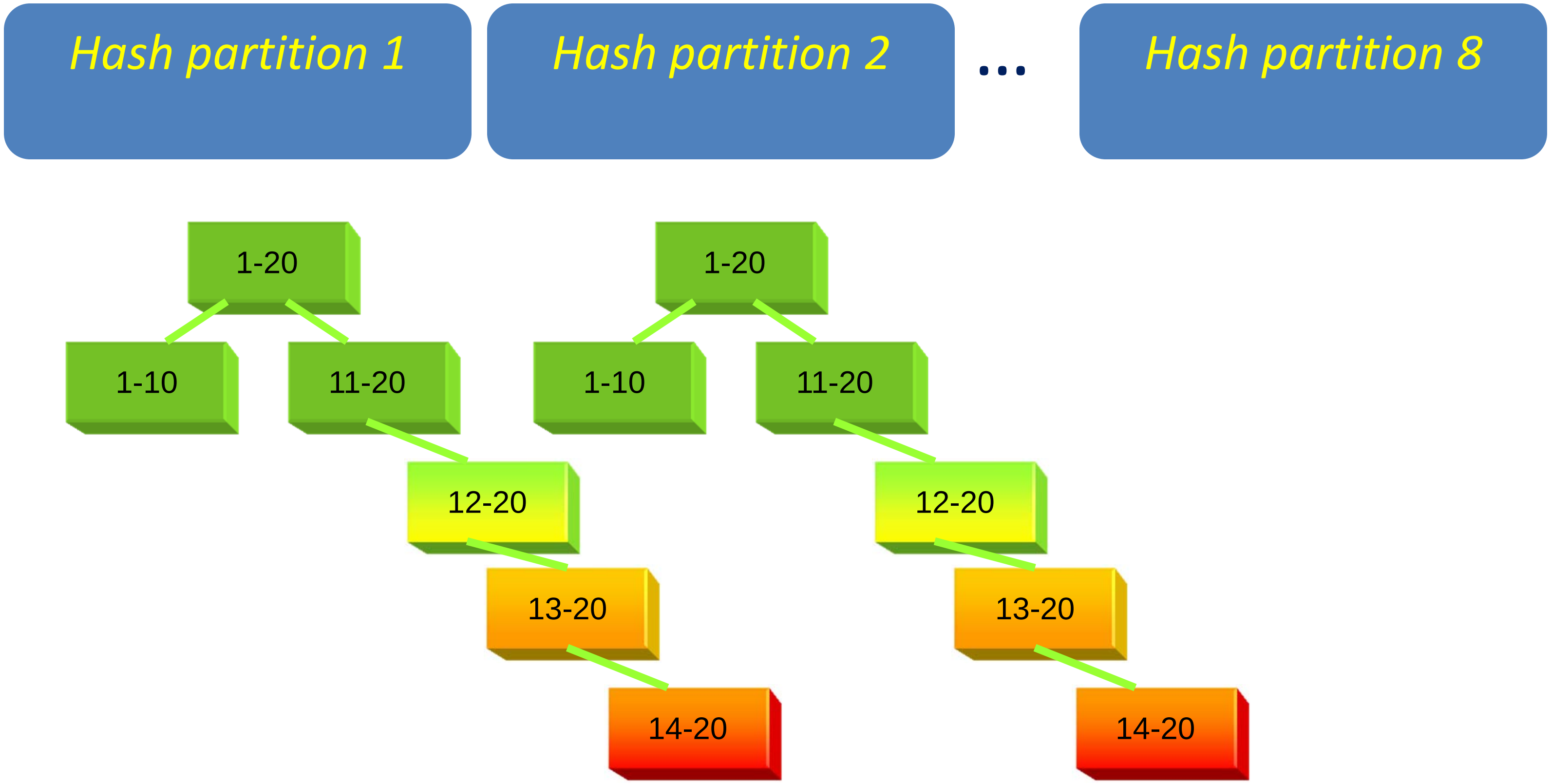
- Index Can be hash-partitioned, regardless of the partitioning status of the table
- Table SALES is un-partitioned; while index is partitioned
- This creates multiple segments for the same index, forcing index blocks to be spread on many branches
- Can be rebuilt:

```
alter index IN_SALES_01 rebuild partition
   <PartName>;
```

- Can be moved, renamed, etc.

```
create index
IN_SALES_01
on SALES
(SALES_TRANS_ID)
global
partition by hash
(SALES_TRANS_ID)
partitions 8
```

Reason



Hash partition 1

Hash partition 2

…

Hash partition 8

1-20

1-10

11-20

12-20

13-20

14-20

1-20

1-10

11-20

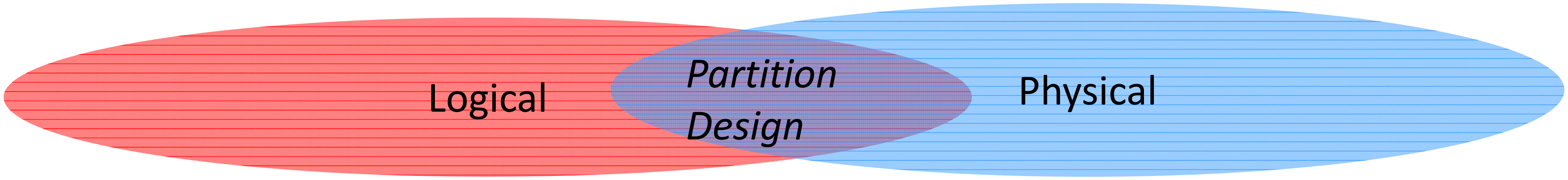
12-20

13-20

14-20

When

- Overlap between Logical Modeling and Physical Design



- Last part of logical design and first part of physical design
- When should partitioning be used
  - In almost all the time for large tables
- There is no advantage in partitioning small tables, right?
  - Wrong. In some cases small tables benefit too

How to Choose

Partitioning

Why? Common Reasons

- Easier Administration:
  - – Smaller chunks are more manageable
  - – Rebuilding indexes partition-by-partition
  - – Data updates, does not need counters
- Performance:
  - – full table scans are actually partition scans
  - – Partitions can be joined to other partitions
  - – Latching

More Important Reasons

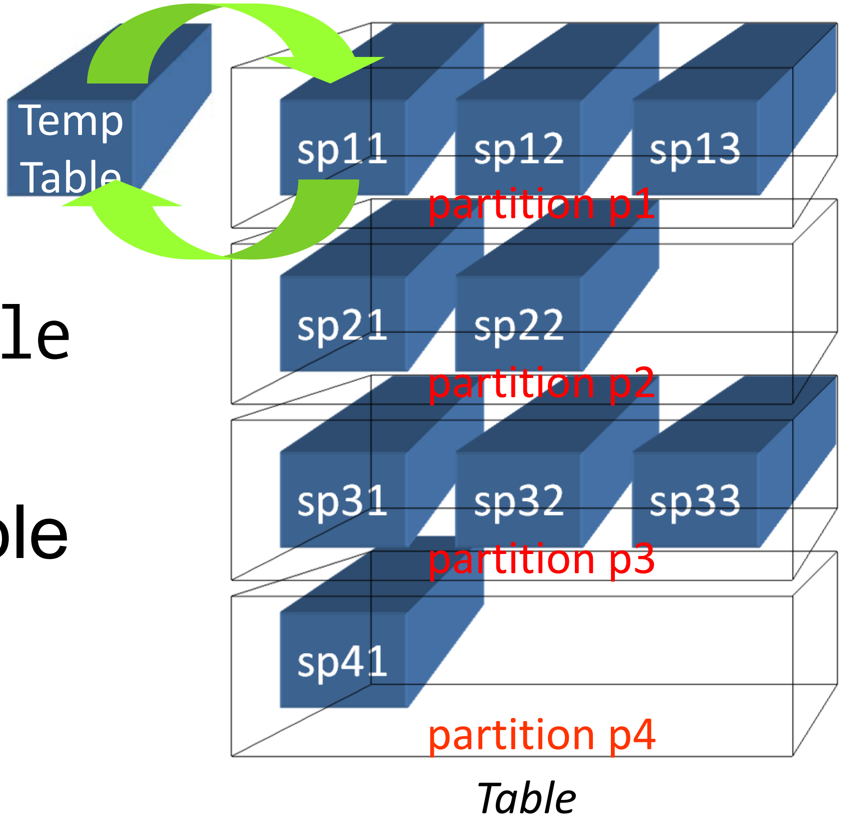
- Data Purging
  - DELETEs are expensive – REDO and UNDO
  - Partition drops are practically free
  - Local indexes need not be rebuilt
- Archival
  - Usual approach: insert into archival table select * from main table
  - Partition exchange
  - Local indexes need not be rebuilt

Materialized Views Refreshes

- Partition Exchange
  - Create a temp table
  - Create Indexes, etc.
  - When done, issue:

```
alter table T1 exchange
  partition sp11 with table
  tmp1;
```

  - Data in TMP1 is available



Backup Efficiency

- When a tablespace is read-only, it does not change and needs only one backup
  - RMAN can skip it in backup
  - Very useful in DW databases
  - Reduces CPU cycles and disk space
- A tablespace can be read only when all partitions in them can be so

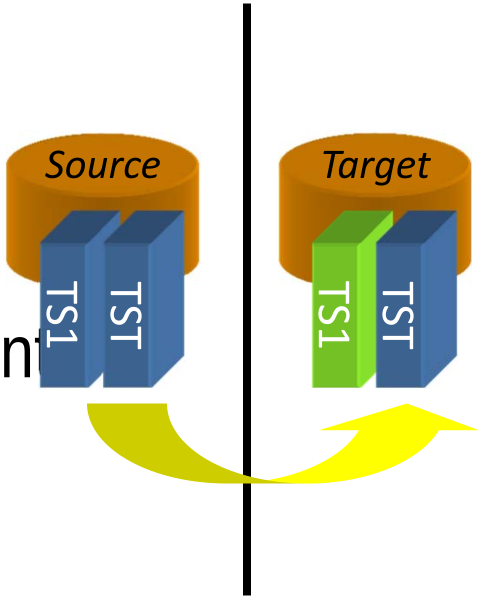
```
SQL> alter tablespace Y08M09 read only;
```

Data Transfer

- Traditional Approach

```
insert into target select * from
  source@dblink
```

- Transportable Tablespace
  - Make it read only
  - Copy the file
  - "Plug in" the file as a new tablespace into the target database
  - Can also be cross-platform



Information Lifecycle Management

- When data is accessed less frequently, that can be moved to a slower and cheaper storage, e.g. from Fiber to SATA
- Two options:

1. Create a tablespace ARC_TS on cheaper disks

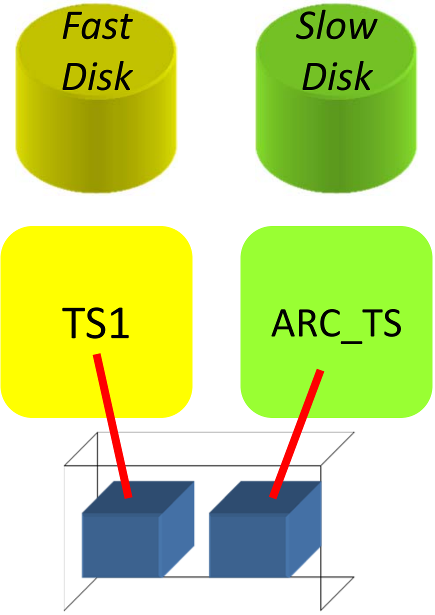
   ```
   ALTER TABLE TableName MOVE
      PARTITION Y07M08 TABLESPACE
      ARC_TS;
   ```

   Reads will be allowed; but not writes

2. ASM Approach

   ```
   ALTER DISKGROUP DROP DISK … ADD DISK
     …
   ```

   Fully available



How to Decide

- First, decide on the objectives of partitioning. Multiple objectives possible
- Objectives
  - Data Purging
  - Data Archival
  - Performance
  - Improving Backups
  - Data Movement
  - Ease of Administration
  - Different Type of Storage

Assign priorities to each of these objectives

Thank You!

My Blog: **arup.blogspot.com**
My Tweeter: **arupnanda**