Partitioning WWWH
What, When, Why & How

Arup Nanda

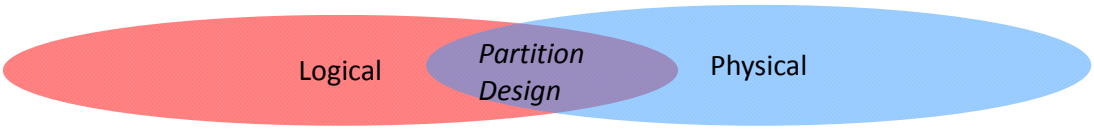
Longtime Oracle DBA

About this Session

- This is not an introduction to partitioning
  - Will not cover syntax
- What type of partitioning
- When to use partitioning
- Why partition something
- How to use partitioning to overcome common challenges
- A complete case study to show how decisions are made

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

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 Causes

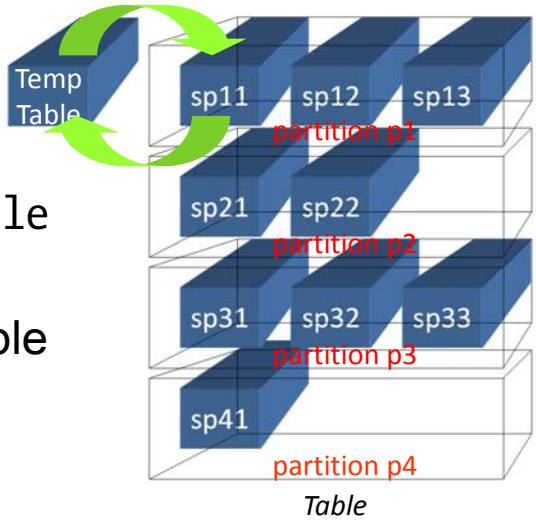
- Data Purging
  - DELETEs are expensive – REDO and UNDO
  - Partition drops are practically free
  - Local indexes need not be rebuilt
- Archival
  - Usual approach: insert into archival table select * from main table
  - Partition exchange
  - Local indexes need not be rebuilt

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

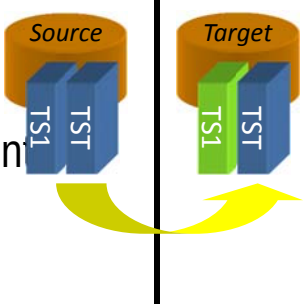
```
SQL> alter tablespace Y08M09 read only;
```


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 in the target database
  - Can also be cross-platform



Information Lifecycle Management

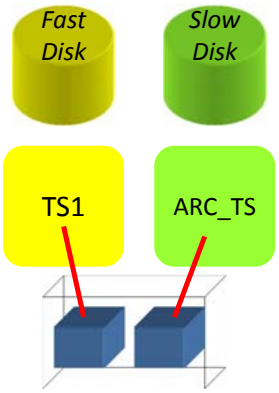
- When data is accessed less frequently, that can be moved to a slower and cheaper storage, e.g. from DMX to SATA
- Two options:
  1. Create a tablespace ARC_TS on cheaper disks

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

     Reads will be allowed; but not writes
  2. ASM Approach

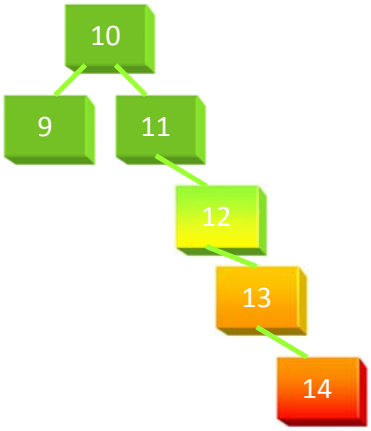
     ```
     ALTER DISKGROUP DROP DISK … ADD DISK
        …
     ```

     Fully available



Index Blocks Too Hot to Handle

- Consider an index on RES_ID, or CK_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



Hash Partitioned Index

- Index Can be hash-partitioned, regardless of the partitioning status of the table

```
create index IN_RES_01 on RES (RES_ID) global
partition by hash (RES_ID)
partitions 8
```

- Table RES 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_RES_01 rebuild partition <PartName>;
```

- Can be moved, renamed, etc.

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

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. RES table. PK – (RES_DT, RES_ID) and part key is (RES_DT)
- If not, try to include the column in PKs
  - E.g. if RES_ID was the PK of RES, can you make it (RES_DT, RES_ID)?
- Ask some hard design questions
  - Do you really need a PK constraint in the DW?

Case Study

- Large Hotel Company
- Fictitious; any resemblance to real or fictional entities is purely coincidental

Background

- Hotel reservations made for *future* dates
- When guests check out, the CHECKOUTS table is populated
- RESERVATIONS has RES_DT
  - Is always in future (up to three years)
- CHECKOUTS has CK_DT
  - Is always present or past.

Thought Process

- Q: How will the tables be purged?
- A: Reservations are deleted 3 months after they are past. They are *not* deleted when cancelled.
  - Checkouts are deleted after 18 months.
- Decision:
  - Since the deletion strategy is based on time, Range Partitioning is the choice with one partition per month.

Column

- Since deletion is based on RES_DT and CK_DT, those columns were chosen as partitioning key for the respective tables
- Scripts:

```
create table reservations (…)
partition by range (res_dt) (
   partition Y08M02 values less than
  (to_date('2008-03-01','yyyy-mm-dd')),
   partition PMAX values less than (MAXVALUE)
)
```

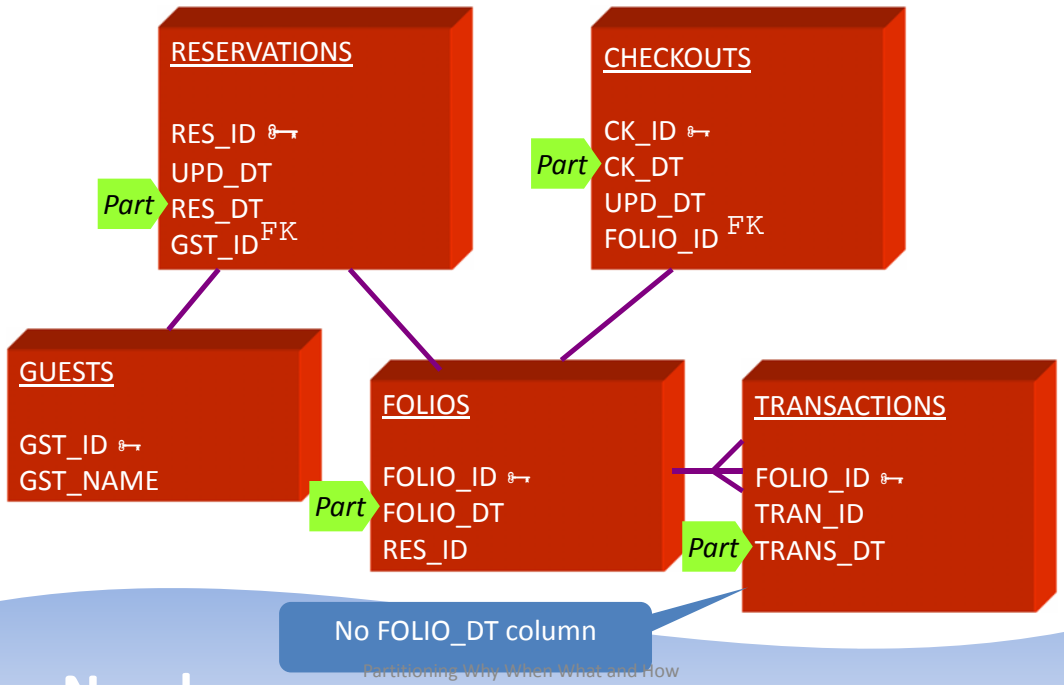

Access Patterns

- Q: Will checkouts within last 18 months be *uniformly* accessed?
  - A: No. Data <= 3 months is heavily accessed. 4-9 months is light; 9+ is rarely accessed.
- Decision:
  - Use Information Lifecycle Management to save storage cost.

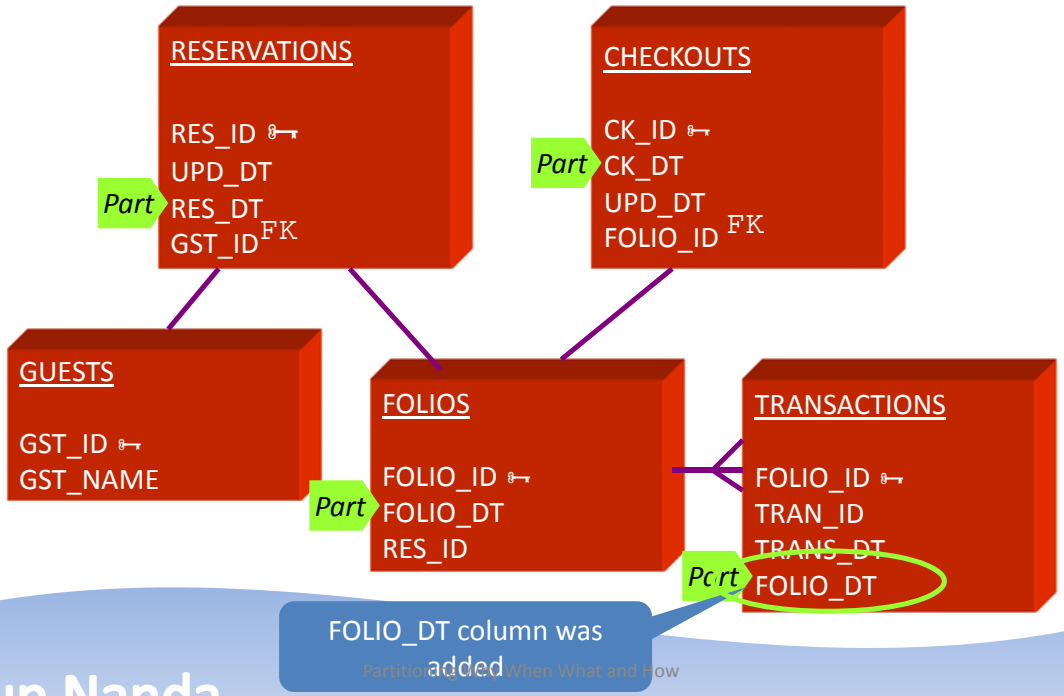
Access Types

- Q: Is it possible that data in past months can change?
  - A: Yes, within 3 months to make adjustments.
- Q: How likely that it will change?
  - A: Infrequent; but it does happen. 3+ months: very rare.
- Q: How about Reservations?
  - A: They can change any time for the future.
- Decision: Make partitions read only.

Partitioning 1st Pass



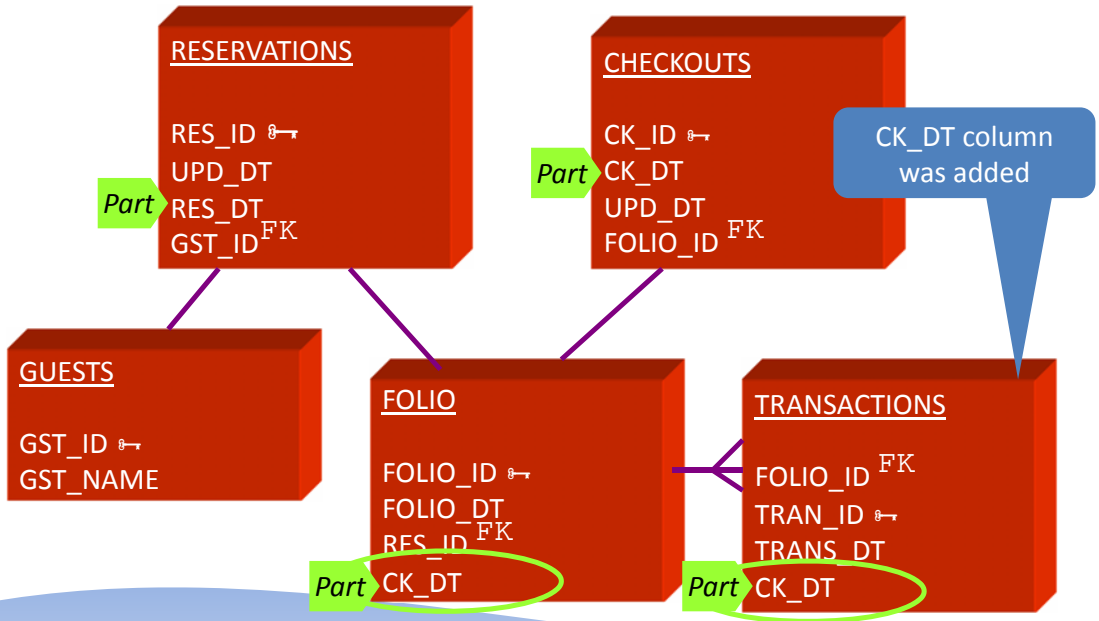
Column Add for Partitioning



Problem

- Purge on CHECKOUTS, FOLIOS and TRANSACTIONS is based on CK_DT, not FOLIO_DT
- FOLIO_DT is the date of creation of the record; CK_DT is updated date
- The difference could be months; so, purging can't be done on FOLIO_DT
- Solution: Partitioning Key = CK_DT
- Add CK_DT to other tables

2nd Pass



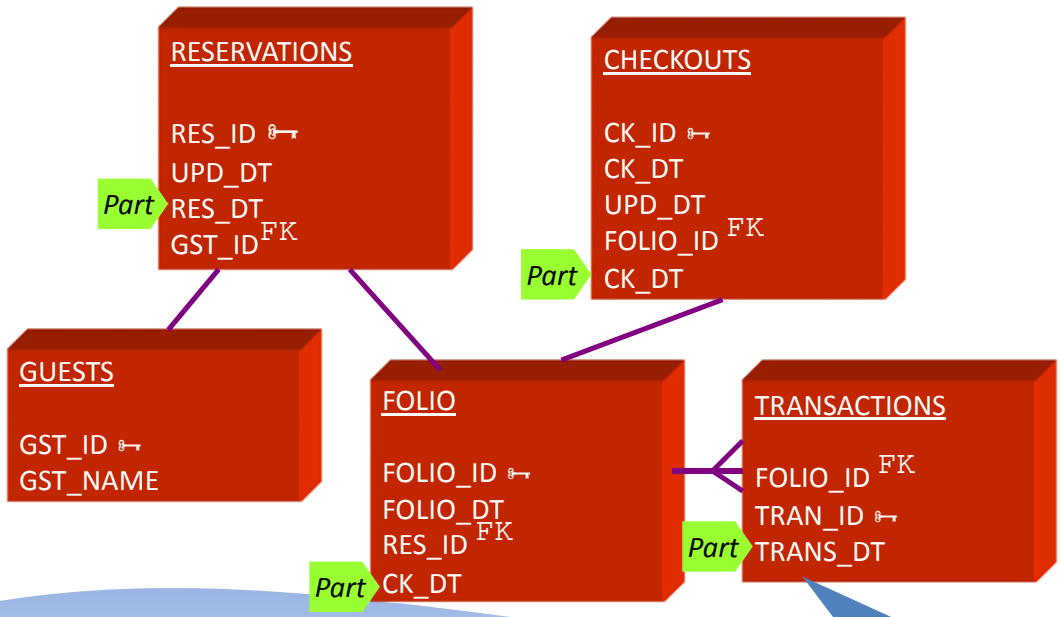
Problems after 2nd Pass

- #1 FOLIOS records created at Check-in
  - CK_DT is updated at Check-out; the record may move to a different partition
  - Decision = Acceptable
- #2 CK_DT will not be known at Check-in; so value will be NULL. Which partition?
  - Decision = not NULL; set to tentative date
  - against Relational Database Puritan Design

Problems, cont..

- #3: TRANS table may have many rows; updating CK_DT may impact negatively
  - Decision: Remove CK_DT from TRANS
  - Partition on TRANS_DT
  - Fact: TRANS_DT <= CK_DT
  - So, when partition SEP08 of CHECKOUTS is dropped, SEP08 partition of TRANSACTIONS can be dropped too
  - Just because part columns are different, purge does not have to different.

3rd Pass



Scenario #1

- Reservation made on Aug 31st for Sep 30th checking out tentatively on Oct 1st
  - Records Created:

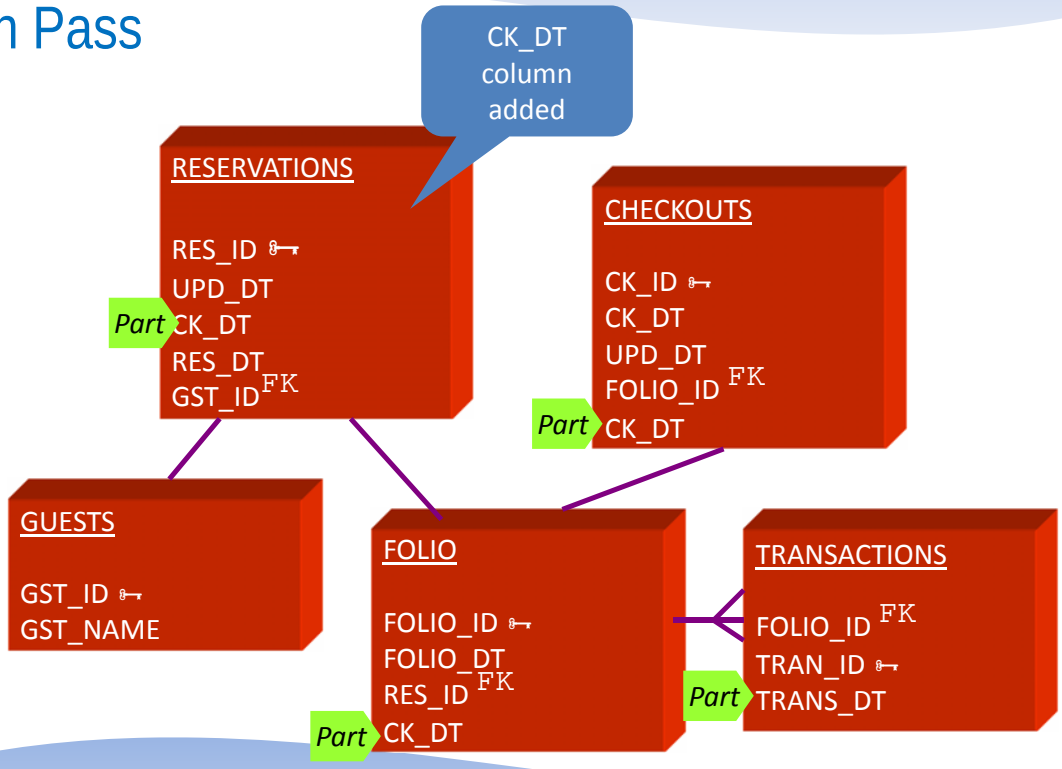
Table	Part Key	UPD_DT	Partition
RESERVATIONS	09/30	08/31	SEP08
FOLIOS	10/01	09/30	OCT08
CHECKOUTS	10/02	10/02	OCT08
TRANSACTIONS	10/02	10/02	OCT08

- Guest checks in on 9/30
- Checks out on Oct 2nd:

CK_DT in RES?

- New Thought:
  - Why not partition RESERVATIONS table by CK_DT as well?
- CK_DT column not present in RES
  - But can be calculated; since we know the number of days of stay.
- Tentative Checkout Date column added

4th Pass



Scenario #1 Modified

- Reservation made on Aug 31st for Sep 30th checking out tentatively on Oct 1st
  - Records Created:

Table	Part Key	UPD_DT	Partition	
RESERVATIONS	10/01	08/31	OCT08	New record

- Guest checks in on 9/30

FOLIOS	10/01	09/30	OCT08	New record

- Checks out on Oct 2nd:

CHECKOUTS	10/02	10/02	OCT08	New record
TRANSACTIONS	10/02	10/02	OCT08	
RESERVATIONS	10/02	10/02	OCT08	Update

Scenario #2

- Guest checks out on Nov 1st, instead of Oct 1st:
  - Records Created:

Table	Part Key	UPD_DT	Partition	
RESERVATIONS	10/01	08/31	OCT08	New record

- Guest checks in on 9/30

FOLIOS	10/01	09/30	OCT08	New record

- Checks out on Nov 1st:

CHECKOUTS	11/01	11/01	NOV08	New record
TRANSACTIONS	11/01	11/01	NOV08	
RESERVATIONS	11/01	11/01	NOV08	Row Migration
FOLIOS	11/01	11/01	NOV08	Row Migration

New Column for Partitioning

- Added a column CK_DT
- Two Options for Populating:
  - Apps populate it (possible since this is still in design)
    - Apps will have to change
    - Guaranteed logic
  - Triggers populate (retrofitting partitioning after the apps are written)
    - No change to apps
    - No guarantee of logic

11g Reference Partitions

- No need to have a new column
- Partitions are defined on Foreign Keys, which follow the parent's partitioning scheme.
- One of the most useful innovations in 11g

```
create table trans (
  trans_id number not null,
  folio_id number not null,
  trans_date date not null,
  amt        number,
  constraint fk_trans_01
    foreign key (folio_id)
    references folios
)
partition by reference
   (fk_trans_01);
```


Non-Range Cases

- GUESTS table is unique:
  - 500 million+ records
  - No purge requirement
  - No logical grouping of data. GUEST_ID is just a meaningless number
  - All dependent tables are accessed concurrently, e.g. GUESTS and ADDRESSES are joined by GUEST_ID
- No meaningful range partitions possible

Hash Partitions

- GUESTS table is hash partitioned on GUEST_ID
- Number of Parts: in such a way that each partition holds 2 million records
- Number of partitions must be a power of 2. So 256 was chosen.
- All dependent tables like ADDRESSES were also partitioned by hash (guest_id)

Hotels Tables

- HOTELS table holds the names of the hotels
- Several dependent tables exist – DESCRIPTIONS, AMENITIES, etc. – all joined to HOTELS by HOTEL_ID
- Partitioning by LIST?

Hotels Table Partitioning

- Requirements:
  - Very small
  - No regular purging needs
  - Mostly static; akin to reference data
  - Can't be read only; since programs update them regularly.
- Decision: No partitioning

Tablespace Decisions

- Partitions of a table can go to
  - Individual tablespaces
  - The same tablespace
- How do you decide?
  - Too many tablespaces ➔ too many datafiles ➔ longer checkpoints

Individual Tablespaces

- Tablespaces named in line with partitions, e.g. RES0809 holds partition Y08M09 of RESERVATION table.
- Easy to make the tablespace READ ONLY
- Easy to backup – backup only once
- Easy to ILM

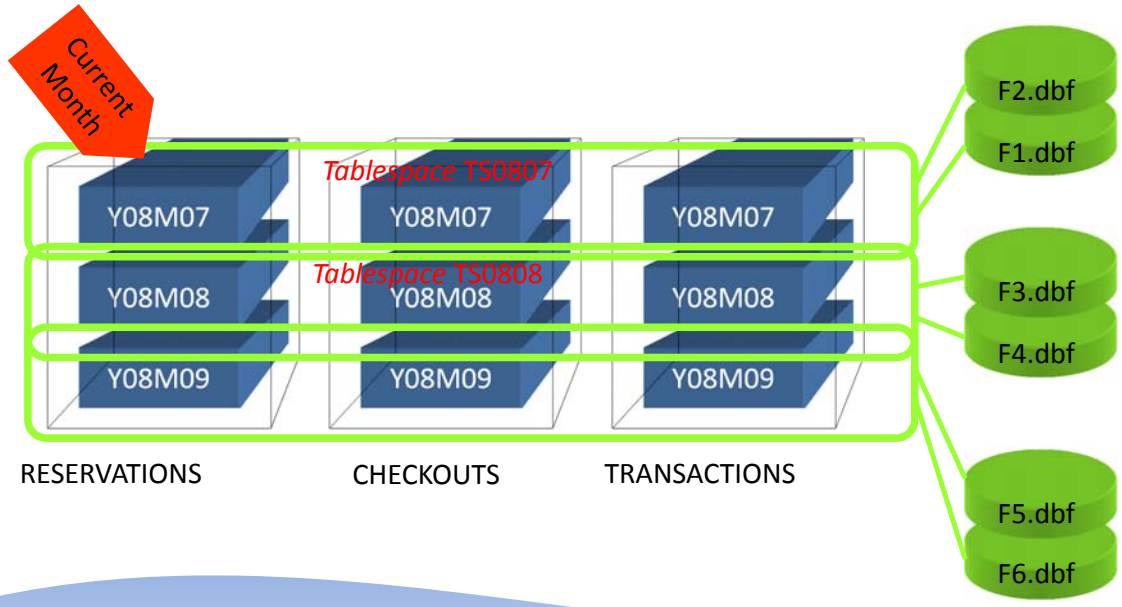
  Move datafiles to lower cost disks

```
ALTER DATABASE DATAFILE '/high_cost/…'
  RENAME TO '/low_cost/…';
```

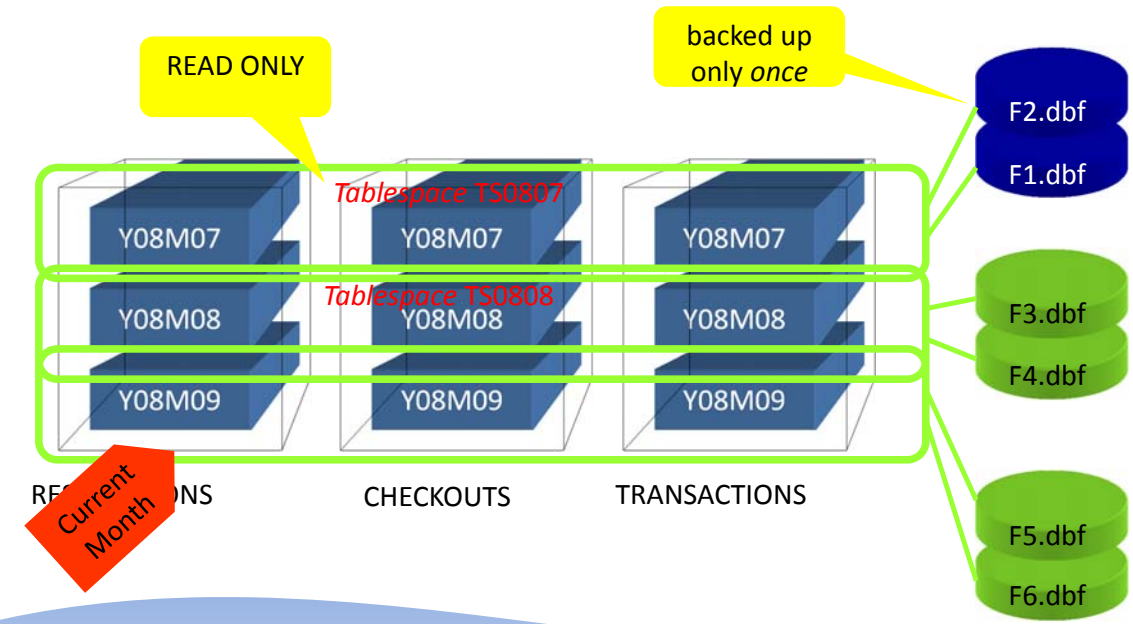

Combined Solution

- Create a tablespace for each period
  - TS0809 for Sep '08
- Contains partitions Y08M09 for all tables – RESERVATIONS, CHECKOUTS, …
- Partitions of the same period for all the tables are usually marked read only
  - If not possible, then this approach fails

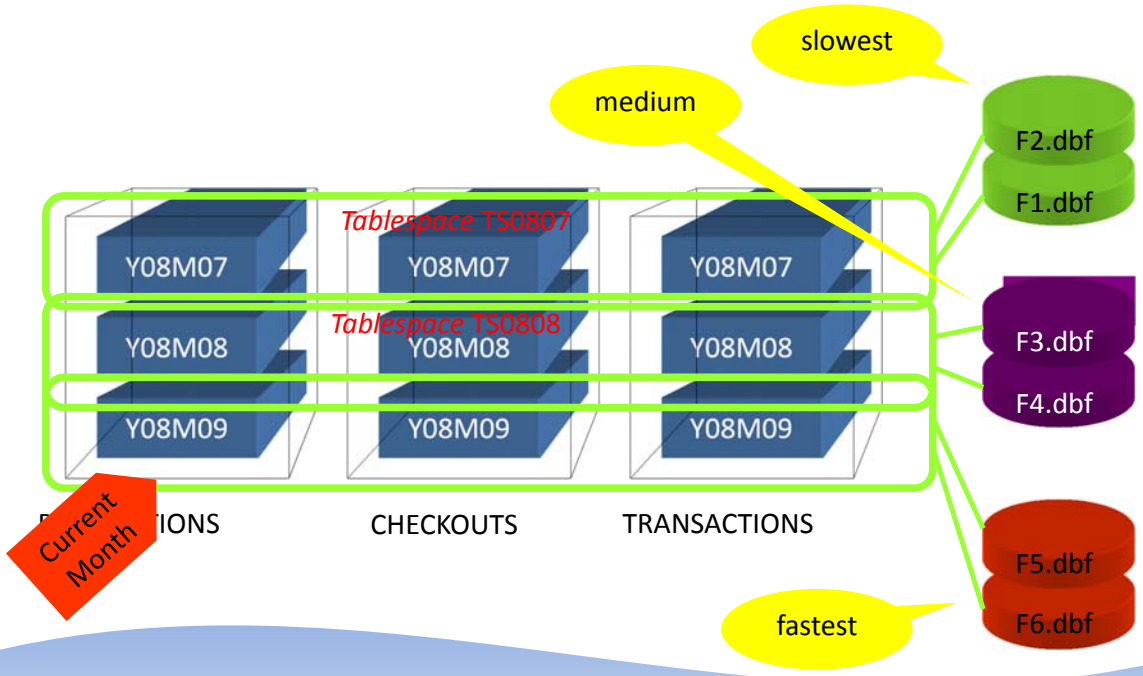
Final Design



Backup



ILM



Partitioning Tips

- List the objectives of partitioning, in the order of priority
- Try to make the same partitioning for all related tables
- Try to introduce new columns
- Avoid Global Indexes

Tips for Choosing Part Key

- Changeable columns do not automatically mean they are not good for part key
- If partition ranges are wide enough, row movement is less likely
- Row movement may not be that terrible, compared to the benefits

Thank You!

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