Partitioning

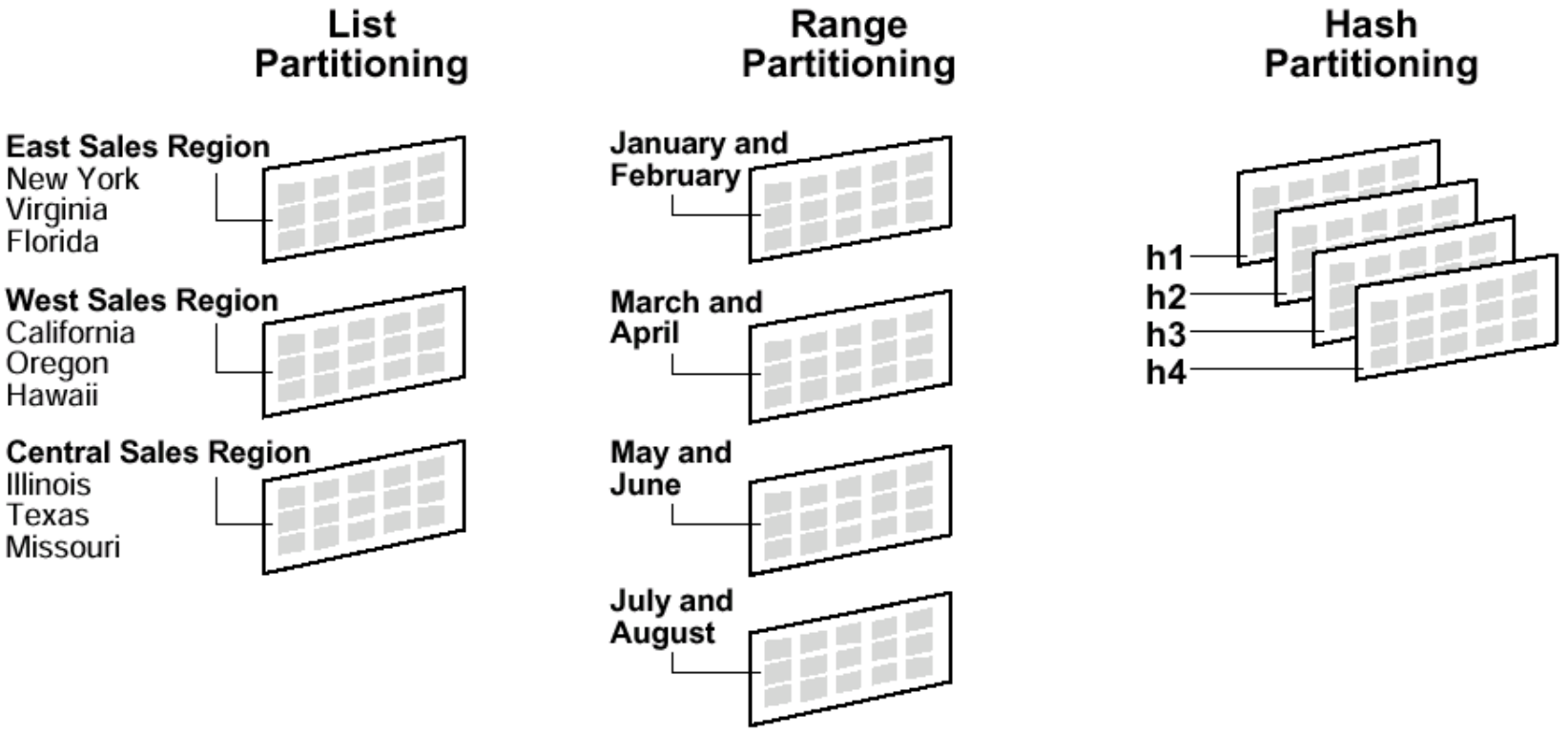
Sukhjit Singh

Partitioned Indexes

Partitioned Indexes

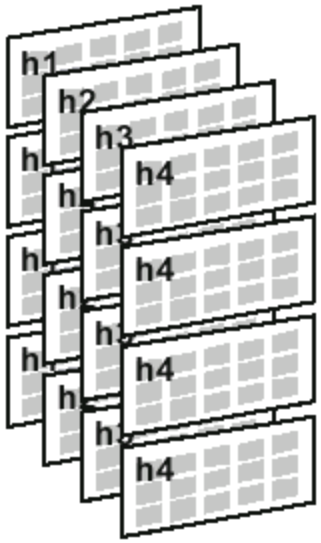
- Partitioning Concepts Review
  - Partitioned Tables
  - Types – Range, Hash, List, Composite
  - Partition Key with 1 or > 1 column
  - Sub-Partitioning Key
- Which table should be partitioned
  - > 2GB in size.
  - Tables consisting of historical data in which data is added in new partition and removed from the oldest to a data warehouse

Partitioned Indexes

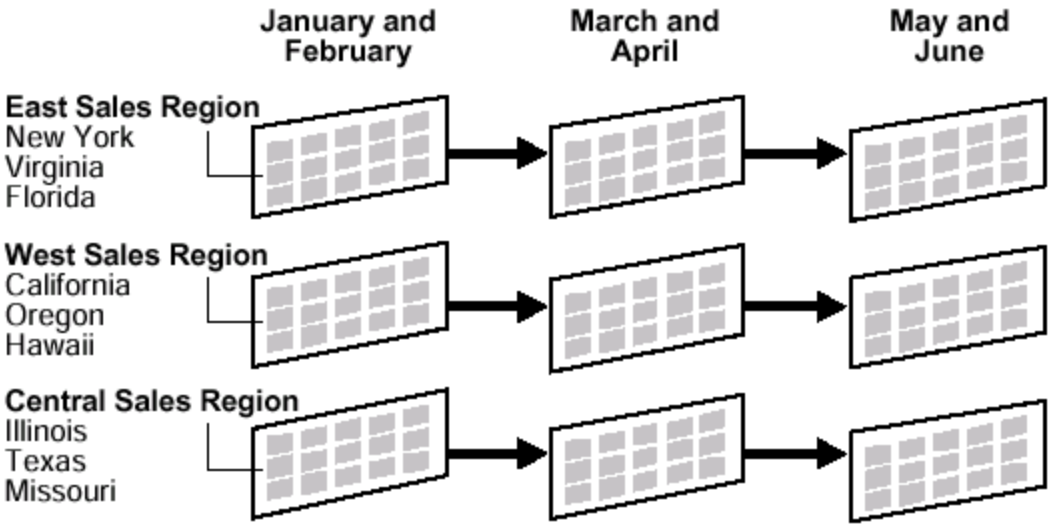


Partitioned Indexes

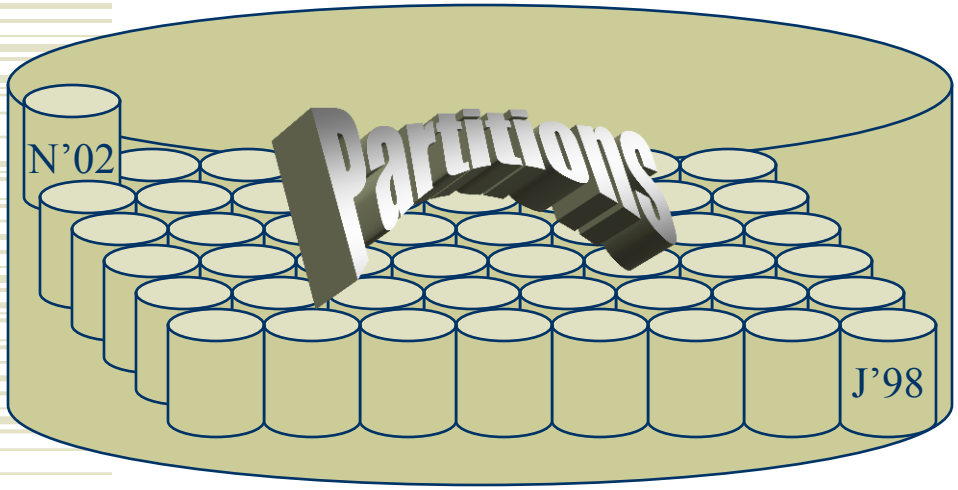
Composite Partitioning
Range-Hash



Composite Partitioning
Range - List



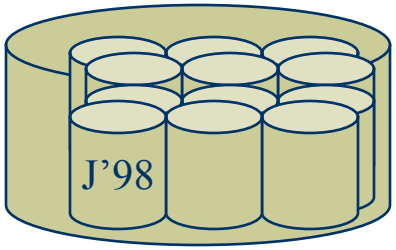
Why use Partitions



Transactional System

Insert a new table partition

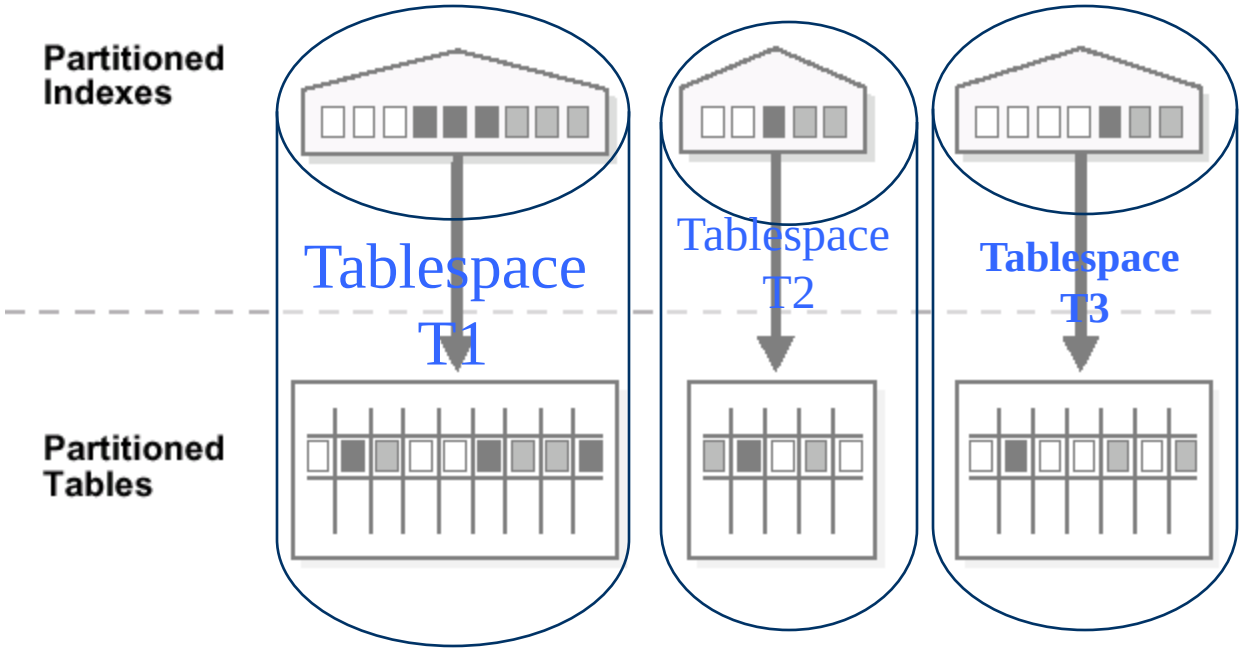
Archive the old partition into the date warehouse



Data Warehouse

Local Partitioned Indexes

- **Local Indexes stay in sync with their respective partitions.**
- **Easy access to relative data in same Tablespace improves performance.**
- **Local Index can be unique or non-unique**



How to create local indexes

Consider the following table:

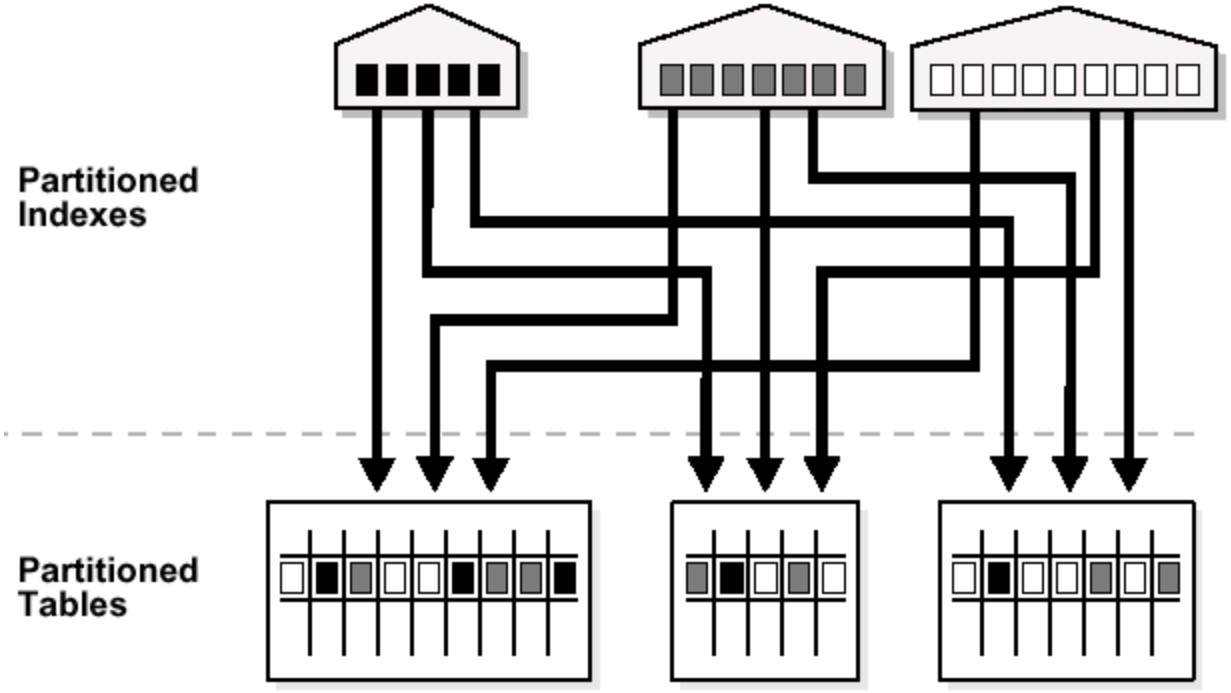
```
CREATE TABLE employees
(employee_id NUMBER(4) NOT NULL,
 last_name VARCHAR2(10),
 department_id NUMBER(2))
PARTITION BY RANGE (department_id)
(PARTITION employees_part1 VALUES LESS THAN (11) TABLESPACE part1,
 PARTITION employees_part2 VALUES LESS THAN (21) TABLESPACE part2,
 PARTITION employees_part3 VALUES LESS THAN (31) TABLESPACE part3);
```

Build a local partitioned index on respective partitions.

```
CREATE INDEX employees_local_idx ON employees (employee_id) LOCAL;
```

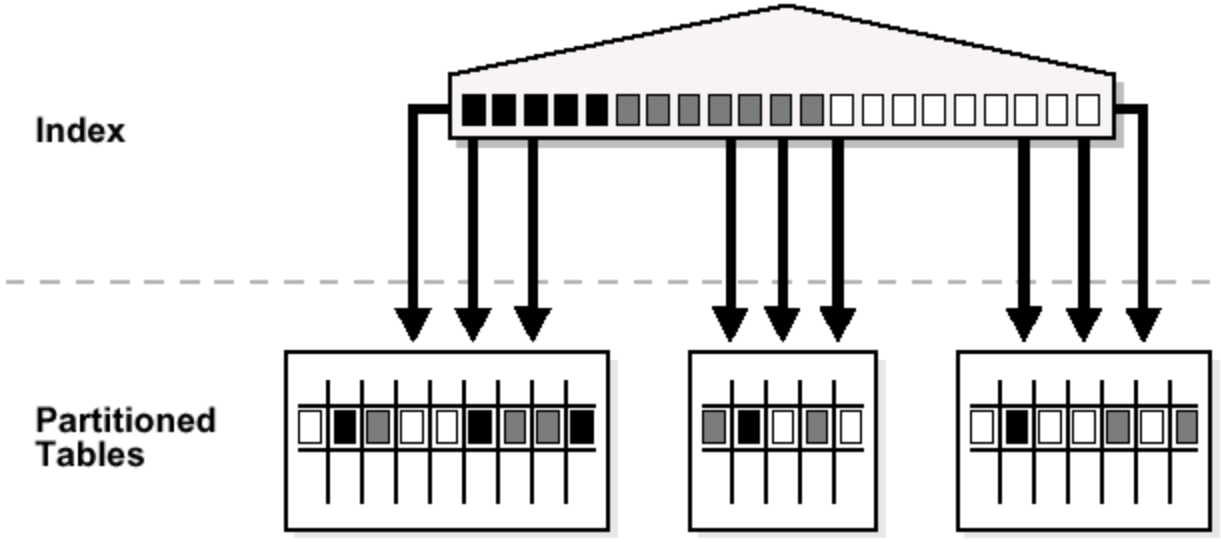
Global Partitioned Indexes

- **Partitioning key for index has relation with partitioning key of Table**
- **Cross Linkage of data leads to poor performance.**



Global Index

- Regular Index that exists above all partitions.
- Behaves the same as a index on regular table.
- Gives poor performance on large tables.



Creating Global Indexes

Consider this Table

```
CREATE TABLE employees
(employee_id NUMBER(4) NOT NULL,
 last_name VARCHAR2(10),
 department_id NUMBER(2))
PARTITION BY RANGE (department_id)
(PARTITION employees_part1 VALUES LESS THAN (11) TABLESPACE part1,
 PARTITION employees_part2 VALUES LESS THAN (21) TABLESPACE part2,
 PARTITION employees_part3 VALUES LESS THAN (31) TABLESPACE part3);
```

Create a Global Index

```
CREATE INDEX employees_global_idx ON employees(employee_id);
CREATE INDEX employees_global_part_idx ON employees(employee_id)
GLOBAL PARTITION BY RANGE(employee_id)
(PARTITION p1 VALUES LESS THAN(5000),
 PARTITION p2 VALUES LESS THAN(MAXVALUE));
```

How to maintain Indexes?

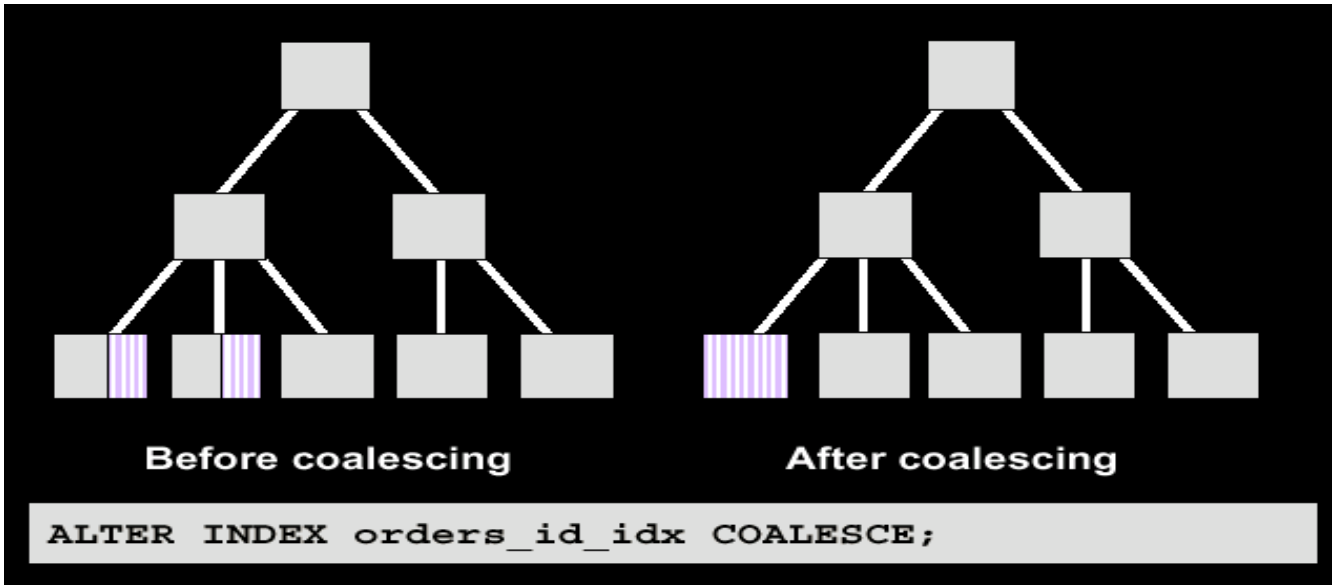
- Rebuild – Will Rebuild the Entire Index
- Add – Adds a partition to an existing index.
- Coalesce – Removes Free Space
- Drop – Removes the index partition
- Exchange – Exchanges the given partition with another table.
- Merge – Combines the two partitions indexed data.
- Move – Moves the partition from one tablespace to another.
- Split – Splits the two partitions to avoid having a very large index. Improves performance.
- Truncate – Empties the data from a given partition.

How to Maintain Indexes?

- Rebuilding indexes can be done with minimal table locking

```
ALTER INDEX orders_id_idx REBUILD ONLINE;
```

Rebuild



Coalesce

How to Maintain Indexes?

- Drop and re-create an index before bulk loads.
- Drop indexes that are infrequently needed and build them when necessary.
- Drop and re-create invalid indexes.

```
DROP INDEX hr.deptartments_name_idx;
```

Dropping Indexes

Check SQL Reference for more information on how to use these commands.

Index Dictionaries

- DBA_Indexes
- DBA_IND_COLUMNS
- DBA_IND_EXPRESSIONS
- V$OBJECT_USAGE

Monitor Usage on Indexes

- To start monitoring the usage of an index

```
ALTER INDEX summit.orders_id_idx
MONITORING USAGE
```

- To stop monitoring the usage of an index

```
ALTER INDEX summit.orders_id_idx
NOMONITORING USAGE
```