Partitioning

Chapter content:

- Partitioning

Table of content

Partitioning

Principle: divide data (table, index) into pieces that can be manipulated independently.

- $\hookrightarrow$ a key feature in datawarehouses!
- several kinds of partitioning (depending on vendors too)
- choice of parameters may be crucial for performance
- transparency (hopefully): queries need not be rewritten

Assets:

✔ more indexing opportunities

✔ easier reorganization/update (monitoring)

Weaknesses:

✘ performance overhead for queries spanning multiple partitions

✘ more tables/complex schema

Partitioning

Principle: divide data (table, index) into pieces that can be manipulated independently.

2 ways to partition a table:

- *vertical partitioning*: split the columns
- *horizontal partitioning*: splits the rows

We only detail partitioning in the DBMS, but also possible to partition data at the application level, or hardware.

Vertical partitioning

Vertical partitioning/Row splitting

splits the attribute of the table into groups. Each group is stored in a separate table.

We already saw a particular (but frequent) case of vertical partitioning:

id	price	amount	code	form	...
1	50	4	AX03F	CASH	
2	60	2	VF67D	VISA	
3	30	1	DS27W	VISA	

id	price	amount
1	50	4
2	60	2
3	30	1

id	code	form	...
1	AX03F	CASH	
2	VF67D	VISA	
3	DS27W	VISA	

Vertical partitioning

- more tables, but fewer columns in each
- different physical devices can be used for each partition
- a view across all partitions allows to restore original rows
- used to store large (BLOBs,LONGs) or seldom-used attributes in separate table (e.g., cheaper storage media)

Assets:

✔ smaller records: more can be loaded *in-memory*

✔ limit reorganization on frequently changing attributes

✔ each partition can be optimized independently

✔ contention (physical storage), parallelism

Weaknesses:

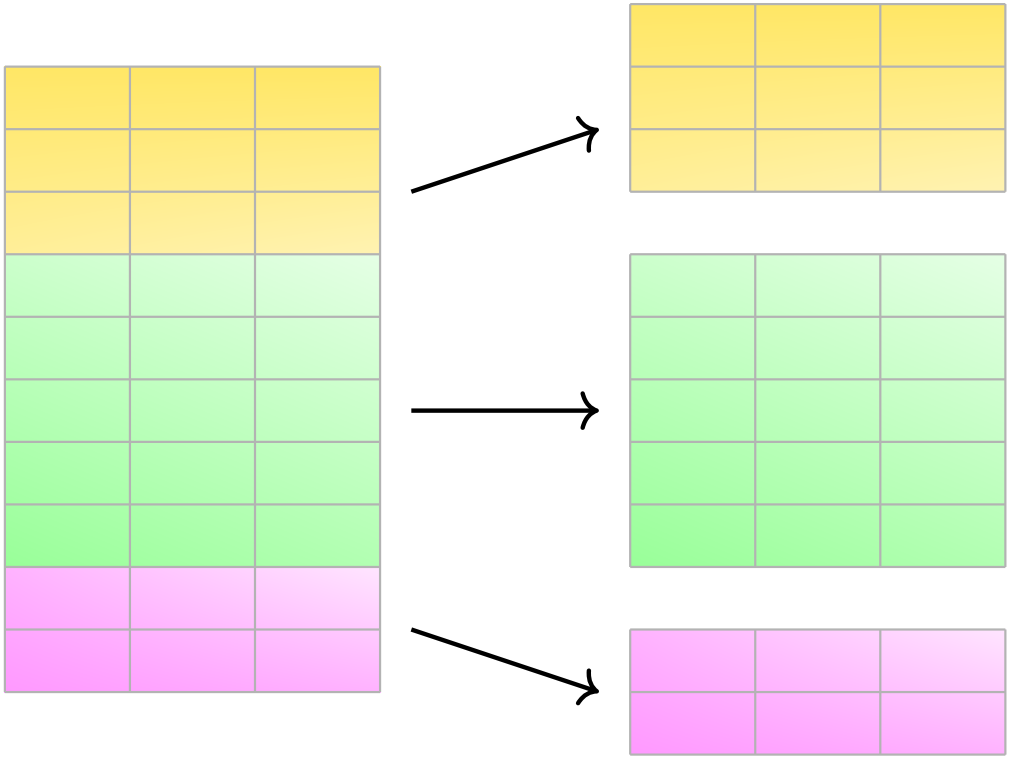
✘ destroys semantic unit: record

✘ joining tables through non-materialized view degrades performance

Horizontal partitioning: principle

Horizontal partitioning

splits the rows of the table into groups based on attribute value. Each group is stored in a separate table.



partitions

Featured in all major DBMS: Oracle, IBM DB2, SQL Server, PostgresQL, Mysql, MariaDB...

- allows to handle very large relations
- may simplify maintenance
- can be used to filter partitions for query optimization (partition pruning)
- allows parallelism

Horizontal partitioning

Horizontal partitioning

splits the rows of the table into groups based on attribute value. Each group is stored in a separate table.

id	price	amount	code	form	...
1	50	4	AX03F	CASH	
2	60	2	VF67D	VISA	
3	30	1	DS27W	VISA	
4	20	3	AP11Z	CASH	

partitioning key

id	price	amount	code	form	...
1	50	4	AX03F	CASH	
2	60	2	VF67D	VISA	

id	price	amount	code	form	...
3	30	1	DS27W	VISA	
4	20	3	AP11Z	CASH	

Horizontal partitioning

splits the rows of the table into groups based on attribute value. Each group is stored in a separate table.

id	price	amount	code	form	...
1	50	4	AX03F	CASH	
2	60	2	VF67D	VISA	
3	30	1	DS27W	VISA	
4	20	3	AP11Z	CASH	

id	price	amount	code	form	...
1	50	4	AX03F	CASH	
4	20	3	AP11Z	CASH	

id	price	amount	code	form	...
3	30	1	DS27W	VISA	
2	60	2	VF67D	VISA	

Horizontal partitioning

. . . each tuple is assigned to exactly one partition depending on value of partitioning key(s).

Oracle syntax (sketch)

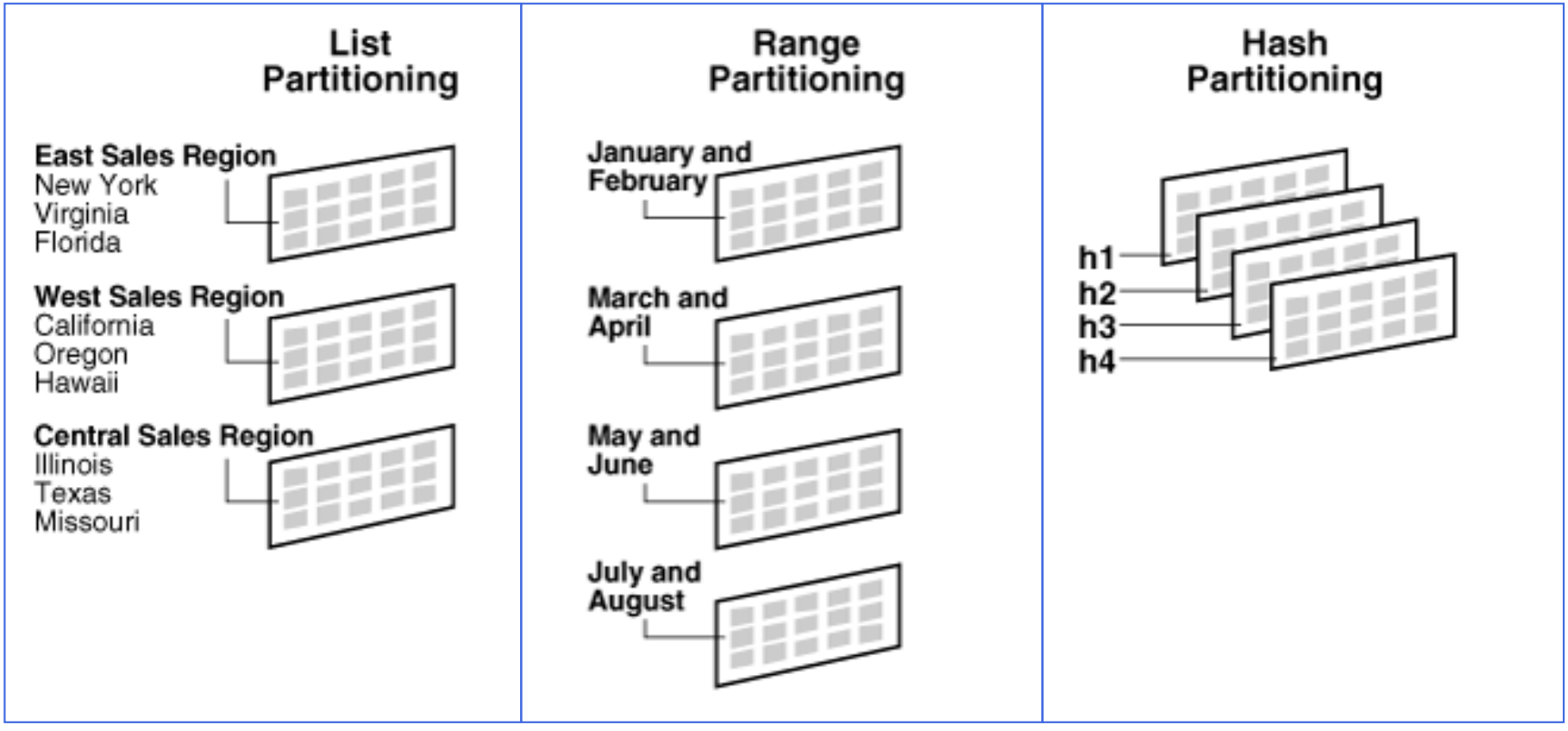
```
PARTITION BY <partition_type> (attribute list) ...
```

partitioning key

in multicolumn keys, column $k+1$ only breaks ties on columns (1,. . . ,k)

Multiple horizontal partitioning options: Oracle

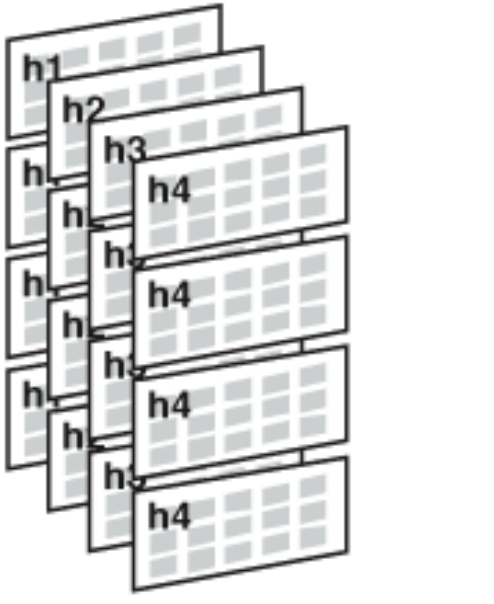
Single level partitioning

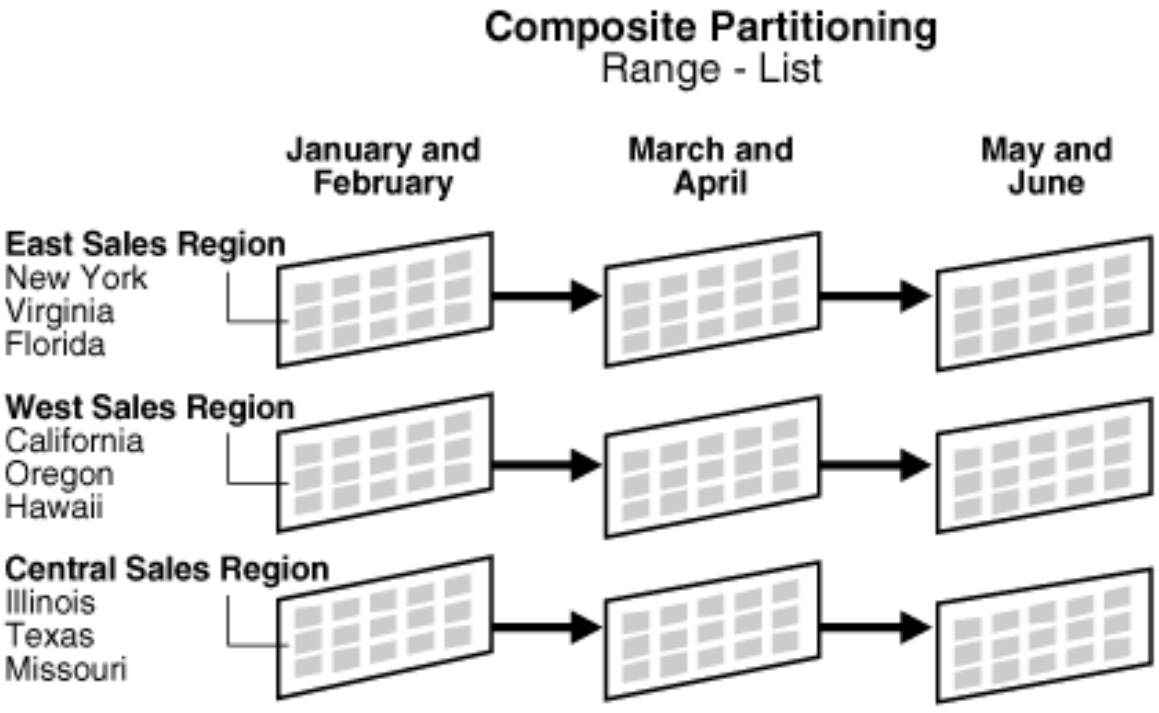


[*Data Warehousing Guide*, Oracle]

Composite partitioning examples: Oracle

Composite partitioning





...

[*Data Warehousing Guide*, Oracle]

RANGE partitioning (Oracle)

PARTITION BY RANGE(column_list) . . . VALUES LESS THAN (value list)

Partitions are specified by upper bound (exclusive).

```
CREATE TABLE sales_range(sales_id int PRIMARY KEY, salesman_id NUMBER(5),
    salesman_name VARCHAR2(30), sales_amount NUMBER(10),sales_date DATE)
  PARTITION BY RANGE(sales_date)
    (PARTITION t21 VALUES LESS THAN(TO_DATE('02/01/2000','DD/MM/YYYY')),
    PARTITION t22 VALUES LESS THAN(TO_DATE('03/01/2000','DD/MM/YYYY')),
    PARTITION t23 VALUES LESS THAN(TO_DATE('04/01/2000','DD/MM/YYYY')),
    PARTITION t24 VALUES LESS THAN(TO_DATE('05/01/2000','DD/MM/YYYY')));
```

MAXVALUE can be used to specify the top range, otherwise: implicit integrity constraint on the table.

[*Data Warehousing Guide*, Oracle]

HASH partitioning (Oracle)

```
CREATE TABLE sales_hash
    (salesman_id NUMBER(5),salesman_name VARCHAR2(30),
    sales_amount NUMBER(10),week_no NUMBER(2))
  PARTITION BY HASH(salesman_id)
  PARTITIONS 4; -- number of partitions
```

For optimal distribution:

- choose adequate partitioning key
  (e.g., column(s) that are almost unique)
- take power of 2 as number of partitions (subpartitions)

LIST partitioning (Oracle)

```
CREATE TABLE sales_list (salesman_id NUMBER(5),
    sales_state VARCHAR2(20), sales_amount NUMBER(10))
  PARTITION BY LIST(sales_state) -- single attribute: no composite column
    (PARTITION sales_west VALUES('California', 'Hawaii'),
    PARTITION sales_east VALUES ('New York', 'Virginia', 'Florida'),
    PARTITION sales_other VALUES(DEFAULT)); -- (optional)
```

Does not support multi-column partition keys.

Additional partitioning options (Oracle)

Interval partitioning

```
CREATE TABLE sales_interv(salesman_id NUMBER(5),salesman_name VARCHAR2(30),
    sales_amount NUMBER(10),sales_date DATE)
  PARTITION BY RANGE(sales_date)
  INTERVAL(NUMTOYMINTERVAL(1, 'MONTH'))
    (PARTITION t21 VALUES LESS THAN(TO_DATE('02/01/2000','DD/MM/YYYY')),
    -- a first range partition must be declared
    PARTITION t22 VALUES LESS THAN(TO_DATE('03/01/2000','DD/MM/YYYY')),
    PARTITION t23 VALUES LESS THAN(TO_DATE('04/01/2000','DD/MM/YYYY')));
-- partitions after 4/01/2000 with interval width of 1 month
```

Partition by reference

```
CREATE TABLE sales_item(sales_id int NOT NULL,
    order_id int NOT NULL, desc VARCHAR2(10),
    CONSTRAINT sales_item_fk
    FOREIGN KEY(sales_id) REFERENCES sales_range(sales_id)
  )
  PARTITION BY REFERENCE(sales_item_fk)
-- requires named referential constraint toward partitioned table
```

[*VLDB and Partitioning Guide*, Oracle]

Additional partitioning options (Oracle)

Partitioning on virtual column

```
CREATE TABLE sales_virt
  (
    sale_id NUMBER(5)
  ,  quantity_sold NUMBER(6) NOT NULL
  ,  unit_price NUMBER(8,2) NOT NULL
  ,  total_amount AS (quantity_sold*unit_price) -- cannot use PLSQL
  )
  PARTITION BY HASH(total_amount)
  PARTITIONS 4;
```

COMPOSITE partitioning (Oracle)

```
CREATE TABLE sales_range_hash(s_productid NUMBER, s_saledate DATE,
  s_custid NUMBER, s_totalprice NUMBER)
PARTITION BY RANGE (s_saledate)
SUBPARTITION BY HASH (s_productid) SUBPARTITIONS 8
  (PARTITION sal99q1 VALUES LESS THAN (TO_DATE('01-APR-1999', 'DD-MON-YYYY')),
  PARTITION sal99q4 VALUES LESS THAN (TO_DATE('01-JAN-2000', 'DD-MON-YYYY')));
```

Partitioning: which and when?

- as a general rule, consider partitioning in Oracle if:
  - table>2Gb, more than 1M rows
  - historic data are read-only
  - multiple media storages
- RANGE partitioning: typically (not exclusively) used on temporal attributes
- HASH partitioning: to balance partition size, typical for non-temporal attributes, or to distribute data on physical storage medias
- COMPOSITE partitioning: depending on applications, if high degree of parallelism required

unbalanced partitions tend to degrade performance

Horizontal partitioning: updates

ROW MOVEMENT clause

Oracle syntax

```
CREATE TABLE...
  PARTITION BY...
  DISABLE/ENABLE ROW MOVEMENT
```

- *row movement enabled*: updating the key may cause a row to move to another partition
- *row movement disabled (default)*: update fails if it would result in row migration

Managing partitions:maintenance

Operations on partitions (or subpartitions)

- ADD: introduce a new partition (or subpartition)
- DROP: removes the partition and delete its content
- TRUNCATE: delete content of a partition

- MERGE: merges 2 partitions into a single one
- SPLIT: redistribute content of partition into 2 new ones

- EXCHANGE: move partition to/from table

and also:

MODIFY, MOVE, RENAME, COALESCE

Add partition

ALTER TABLE <tablename> ADD PARTITION ...

```
ALTER TABLE sales_range
  ADD PARTITION  jan00 VALUES LESS THAN ('01-FEB-2000')
  TABLESPACE tsx;
```

```
ALTER TABLE sales_range_hash MODIFY PARTITION  sal99q1
  ADD SUBPARTITION subpart9 TABLESPACE ts99q1;
-- the rows within partition sal99q1 are re-hashed
```

operations depend on partition type.

DROP partition

ALTER TABLE <tablename> DROP PARTITION ...

Fastest way to delete large volumes of data.

```
ALTER TABLE sales_range  DROP PARTITION  jan00;
-- take special care for (global) indexes and integrity constraints:

-- indexes: add 'UPDATE INDEXES' clause
-- or 'ALTER INDEX sales_idx REBUILD'
-- or first 'DELETE FROM sales_range PARTITION (jan00)' before DROP

-- integrity constraints: disable them, or first DELETE
```

Maintaining indexes during DROP may degrade performance.

TRUNCATE: similar to DROP, but only rows are deleted;
the partition (emptied) remains.

operations depend on partition type.

SPLIT/MERGE partition(s)

ALTER TABLE <tablename> SPLIT PARTITION ...

```
ALTER TABLE sales_range
  SPLIT PARTITION  t21 AT (TO_DATE('01/01/2000','DD/MM/YYYY'))
  INTO (PARTITION t21a, Partition t21b);
```

ALTER TABLE <tablename> MERGE PARTITIONS ...

```
ALTER TABLE sales_range
  MERGE PARTITIONS  t21a, t21b INTO PARTITION t21;
```

operations depend on partition type.

Partition exchange

Oracle syntax

```
ALTER TABLE <tname>
   EXCHANGE PARTITION <pname>
   WITH TABLE <tname2>
   [INCLUDING/EXCLUDING INDEXES]  --> exchange index(es)
   [WITH/WITHOUT VALIDATION]      ----> check rows suitable for partition
```

- Swaps a table and a partition.
- table and partition must have same schema, FK constraint, indexes
- extremely useful in ETL.

Typically, organize new data; indexes, constraints... in dedicated table, then integrate new data using partition exchange.

```
CREATE TABLE sales_apr_src(sales_id int PRIMARY KEY, salesman_id NUMBER(5),
    salesman_name VARCHAR2(30), sales_amount NUMBER(10),sales_date DATE);

INSERT INTO sales_apr (1,2,'T. Elliot',23,TO_DATE('04/21/2000','DD/MM/YYYY'));

ALTER TABLE sales_range EXCHANGE PARTITION t24 WITH TABLE sales_apr;
```

Partitioning benefits

(Reminder)

- allows to handle very large relations
- allows parallelism
- may simplify maintenance ✓ (ETL,archiving. . .)
- allows to filter partitions for query optimization (partition pruning)

Partition pruning

Principle: *"Do not scan partitions where there can be no matching values" when the query filters values based on partitioning column.*

Optimizer selects relevant partitions, scan partition instead of whole table.

Predicates supported:

RANGE, LIST	*IN, =, LIKE, range (<,...)*
HASH	*IN, =*

Oracle distinguishes

- static partition pruning: optimizer determines partition at compile time (no subquery, static predicate...)
- dynamic partition pruning: for statements that use subqueries...

Partition-wise joins

Parallelism within a query (principle): *split the query in multiple subqueries over distinct parts of the data; process subqueries in parallel.*

Aim: speedup join processing by minimizing data exchange between parallel execution servers (reduces both communication and memory).

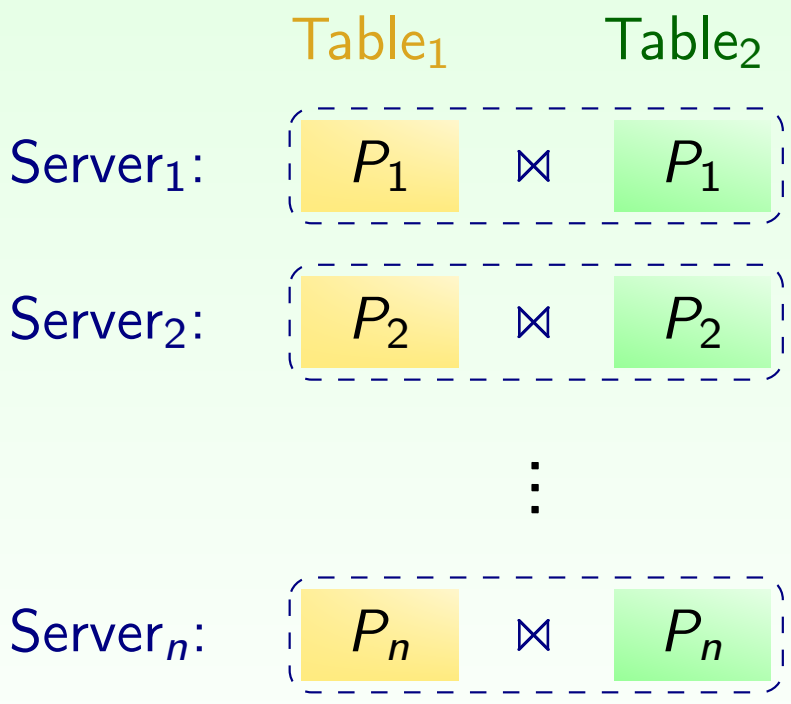
Partition-wise joins can be:

- *"full"* : the 2 tables are partitioned on join key
- *"partial"* : 1 of the tables must be partitioned on join key

Full partitions-wise joins

Joined tables must be partitioned on join key, with same partition criterias (for hash: same # partitions).

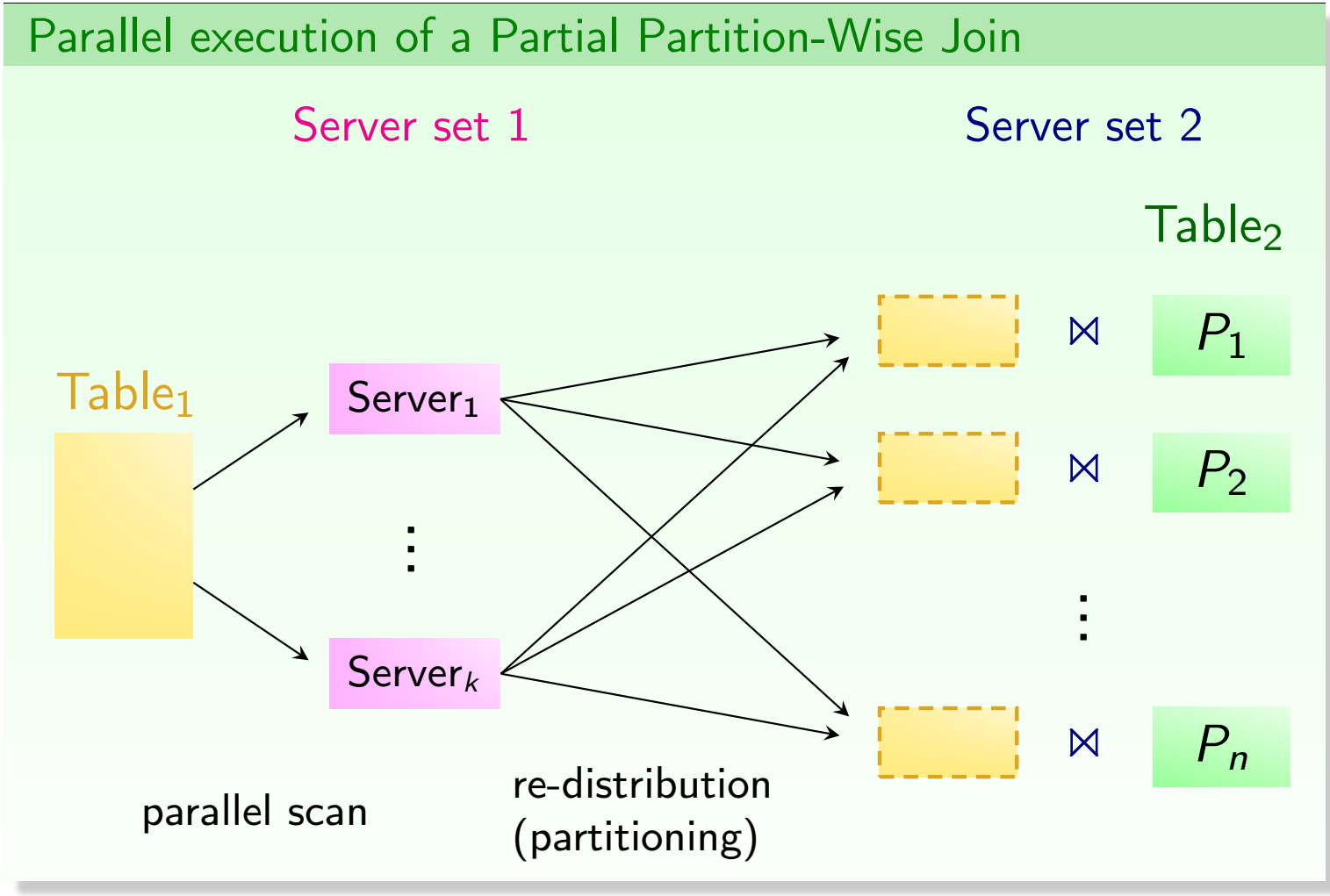
Parallel execution of a Full Partition-Wise Join



- Can be processed serially or in parallel
- Reference partitioning is an easy way to enable full partition-wise joins.

Partial partition-wise joins

Second table needs not be partitioned on join key, as it will be dynamically (re)partitioned based on the reference table partition.



Can be processed in parallel only.

References

Vendors' doc:

VLDB and Partitioning Guide, Oracle (inspired the whole section):
https://docs.oracle.com/database/121/VLDBG/title.htm

https://msdn.microsoft.com/en-us/library/ms190787.aspx

http://www-01.ibm.com/support/knowledgecenter/SSEPGG_9.7.0/com.ibm.db2.luw.admin.partition.doc/doc/c0021560.html

http://www.postgresql.org/docs/9.4/static/ddl-partitioning.html

https://mariadb.com/kb/en/mariadb/using-connect-partitioning-and-sharding/