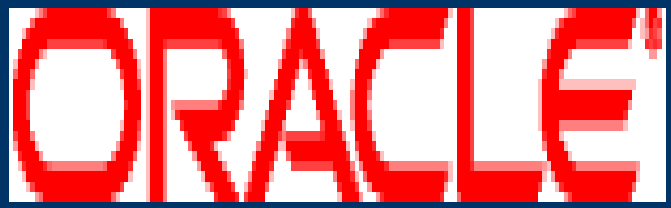
Partizionamento di tabelle ed indici in



- Introduzione al partizionamento
- Metodi di partizionamento
- Partizionamento di indici
- Performance

Introduzione al partizionamento

- Definizione di partizionamento
- Caratteristiche del partizionamento in Oracle
- Vantaggi del partizionamento
- Chiave di partizionamento

DEFINIZIONE

PARTIZIONAMENTO:

Procedura con la quale si decompongono tabelle ed indici di grosse dimensioni in parti più piccole e maneggevoli chiamate partizioni

Caratteristiche del partizionamento

- Gli indici e le tabelle possono essere partizionate solo se non fanno parte di un cluster
- Gli statement SQL e DML non devono essere modificate per accedere alle tabelle partizionate
- Dopo che le partizioni sono state definite, gli statement DDL possono accedere e manipolare individualmente le partizioni puttosto che intere tabelle o indici

Caratteristiche del partizionamento

- Per esmpio si può effettuare l'ALTER di una singola partizione, mentre la SELECT si riferisce all'intera tabella
- Le partizioni sono trasparenti a livello applicativo (quindi all'utente finale)
- Ogni partizione di una tabella o indice ha la stessa struttura in termini di attributi logici dell'oggetto (nomi colonne, tipi di dati, ...)

Caratteristiche del partizionamento

- Ogni partizione può avere diversi attributi fisici come PCTFREE (percentuale di spazio lasciato libero nel blocco per futuri aggiornamenti), PCTUSED (percentuale sotto la quale deve scendere il blocco per successivi inserimenti) e TABLESPACE
- Il partizionamento è usato per molti differenti tipi di aplicazioni, in particolare applicazioni che gestiscono un gran numero di dati

Caratteristiche del partizionamento

- Tutte le partizioni di un oggetto partizionato devono risiedere nelle TABLESPACE di un singolo formato di blocco
- I sistemi OLTP hanno miglioramenti in maneggevolezza e disponibilità
- I sistemi di data warehouse hanno benefici in performanza e maneggevolezza

Vantaggi del partizionamento

- Migliora la granularità di molte operazioni sul database (backup, recovery, ...)
- Miglioramento delle performance delle query (partition pruning)
- DDL meno invasive
- Operazioni di manutenzione del database più semplici perchè eseguito solo su parti dell'oggetto

Vantaggi del partizionamento

- Miglioramento della disponibilità per i database mission critical (database che hanno come fattore determinante la disponibilità del servizio)
- Tutto avviene trasparentemente all'utente finale ed alle varie applicazioni che accedono al database

Chiave di partizionamento

- Ogni riga in una partizione è inambiguamente assegnata ad una singola partizione
- La chiave di partizionamento è un insieme di una o più colonne che determinano la partizione di appartenenza di ogni riga
- In base ai valori della chiave di partizionamento Oracle supporta in modo automatico tutte le operazioni SQL sulla partizione che contiene la riga interessata

Chiave di partizionamento

- Non può contenere una colonna di tipo ROWID
- Non può contenere colonne che contengono valori NULL
- Composta da una lista da 1 a 16 colonne
- Le tabelle possono essere partizionate in più di 64000 separate partizioni. Ma non può contenere colonne con LONG o LONG ROW, invece può contenere tipi CLOB o BLOB

Metodi di partizionamento

- Range Partitioning
- List Partitioning
- Hash Partitioning
- Composite Partitioning:
  - Range – Hash
  - Range – List (presente da Oracle 9.2)

Range Partitioning / 1

- Metodo più comune utilizzato per i dati quando c'è un range logico (*es. date, numeri seriali, ...*)
- Mappa i dati nella rispettiva partizione a condizione che la *partition key* appartenga ad un certo range di valori

Range Partitioning / 2

- Vanno considerate le seguenti regole:

1. Ogni partizione ha una clausola del tipo VALUE LESS THAN, che specifica un *upper-bound* non inclusivo => un valore uguale o maggiore a questo andrà collocato in una partizione successiva.

Range Partitioning / 3

1. Tutte le partizioni, ad eccezione della prima, hanno un *lower-bound* implicito specificato nella partizione precedente

5. È possibile definire un MAXVALUE per la partizione più grande. È un valore oltre il quale nessun valore è ammissibile.

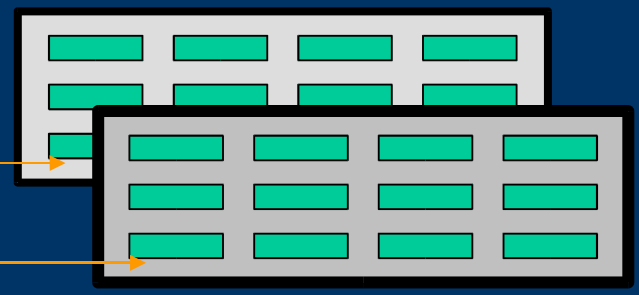
Range Partitioning / 4 - esempio

```
CREATE TABLE Vendite
(CodVendita       NUMBER(5),
 ……...            ……..
 Data_Vendita     DATE)
PARTITION BY RANGE (Data_Vendita)
(PARTITION Vendite_G&F VALUES LESS THAN ('01-MAR-2004')
        TABLESPACE P1,
 PARTITION Vendite_M&A VALUES LESS THAN ('01-MAY-2004')
        TABLESPACE P2);
```

Partition Key

P1: Gennaio & Febbraio

P2: Marzo & Aprile



List Partitioning / 1

- Concede controllo esplicito all'utente mediante definizione di un insieme discreto di valori
- Mappa i dati nella rispettiva partizione a condizione che il valore della *partition key* sia contenuta tra i valori specificati per la partizione

List Partitioning / 2

- Non supporta *partition key* multicolonne, altrimenti si creerebbe confusione all'atto della mappatura nella partizione corretta
- In *Oracle 9.2* è stata aggiunta la partizione di "DEFAULT": evita che le tuple che non hanno partizione di destinazione generino errore

List Partitioning / 3 - esempio

```
CREATE TABLE Vendite
(CodVendita          NUMBER(5),
 ……..                ……..
 Regione_Vendita  VARCHAR(20)
PARTITION BY LIST (Regione_Vendita)
(PARTITION NordOvest VALUES (’Lombardia’,’Piemonte’),
 PARTITION NordEst VALUES (’Veneto’,’Friuli’))
```

Hash Partitioning / 1

- Utilizzato quando i dati non sono partizionabili con il metodo range o list
- Semplice da implementare
- Sfrutta una semplice sintassi

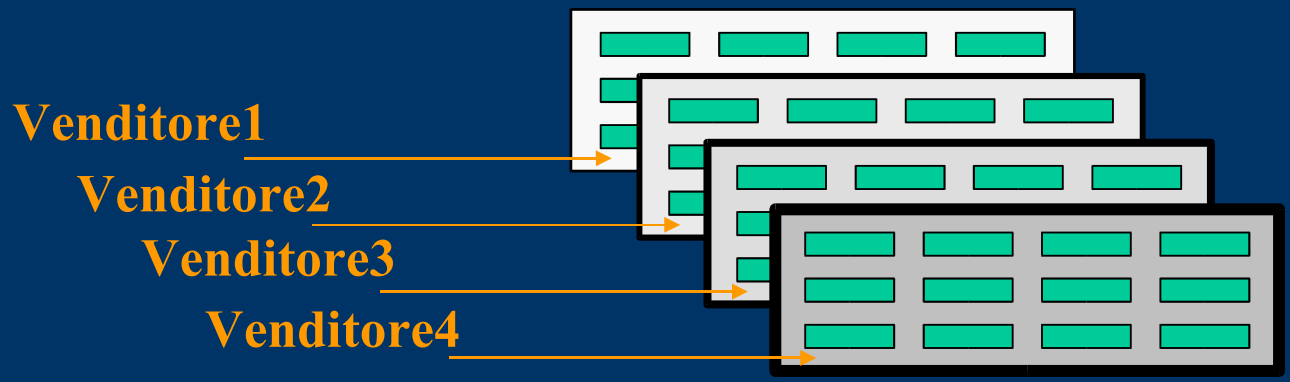
Hash Partitioning / 2

É preferibile nei casi in cui:

- Non si conoscono i dati da mappare
- C’è il rischio che le dimensioni delle partizioni differiscano sostanzialmente
  - poco bilanciamento -
- Il partizionamento range causerebbe ai dati una clusterizzazione insoddisfacente

Hash Partitioning / 3 - esempio

```
CREATE TABLE Vendite
(CodVendita          NUMBER(5),
 ……..                ……..
 Cod_Venditore       NUMBER(5))
PARTITION BY HASH (Cod_Venditore)
PARTITIONS 4
STORE IN (’Venditore1’, ’Venditore2’, ’Venditore3’, ’Venditore4’)
```



Composite Partitioning / 1

- Partiziona i dati usando il metodo range
- Per ogni partizione ottenuta si generano sottopartizioni usando i metodi hash o list

Composite Partitioning / 2

- La scelta su quale metodo abbinare al range dipende dai benefici che si vogliono ottenere dal partizionamento:
  1. Uso list se desidero abbinare, alla gestibilità delle partizioni range, il controllo esplicito delle sottopartizioni
  1. Uso hash se invece desidero ottenere maggiore bilanciamento e parallelismo sulle sottopartizioni

Composite Partitioning / 3 – esempio Range - Hash

```
CREATE TABLE Vendite
(CodVendita          NUMBER(5),
 ……..                ……..
 Cod_Venditore       NUMBER(5)
 Data_Vendita        DATE)
PARTITION BY RANGE (Data_Vendita)
SUBPARTITION BY HASH (Cod_Venditore)
SUBPARTITION TEMPLATE(
  SUBPARTITION sp1 TABLESPACE Venditore1,
  SUBPARTITION sp2 TABLESPACE Venditore2,
  SUBPARTITION sp3 TABLESPACE Venditore3,
  SUBPARTITION sp4 TABLESPACE Venditore4)
(PARTITION Vendite_G&F VALUES LESS THAN ('01-MAR-2004'),
 PARTITION Vendite_M&A VALUES LESS THAN ('01-MAY-2004')))
```

Definizione attributi

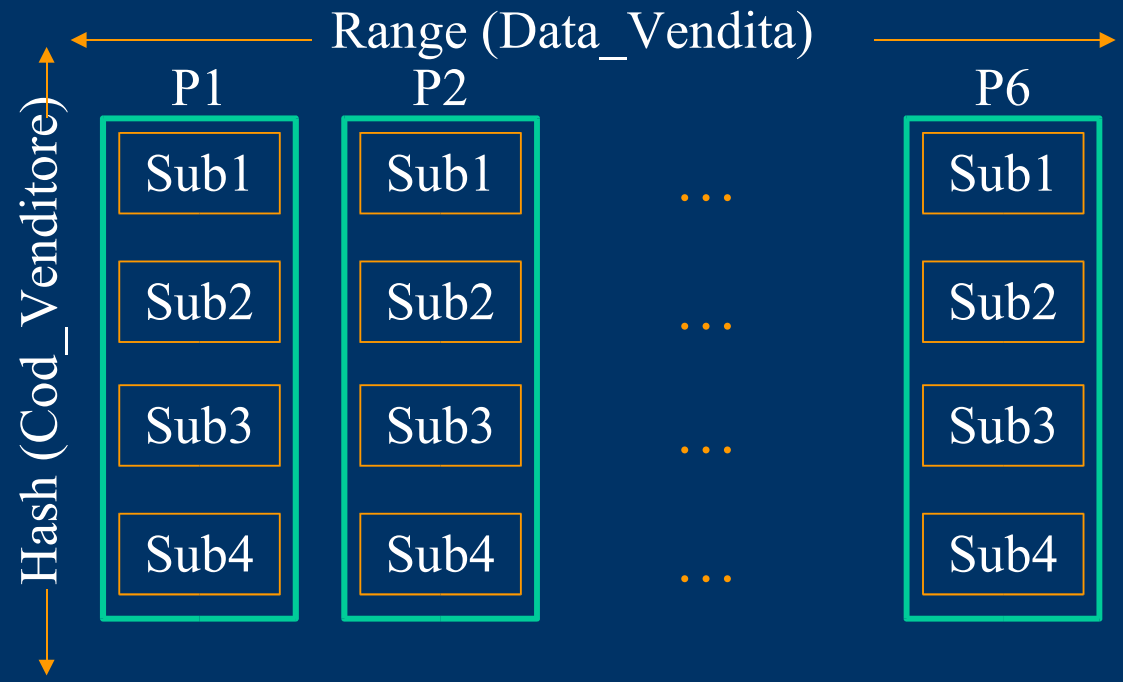
Definizione modalità utilizzate

Definizione sottopartizioni

Definizione partizioni

Composite Partitioning / 4 – esempio Range - Hash

- La tabella precedente può essere letta nel seguente modo grafico:



Composite Partitioning / 5 – esempio Range - List

```
CREATE TABLE Vendite
(CodVendita          NUMBER(5),
 ……..                ……..
 Regione_Vendita     VARCHAR(20)
 Data_Vendita        DATE)
PARTITION BY RANGE (Data_Vendita)
SUBPARTITION BY LIST (Regione_Vendita)
SUBPARTITION TEMPLATE(
   SUBPARTITION NordOvest     VALUES ('Lombardia','Piemonte')
                              TABLESPACE Ts1,
   SUBPARTITION NordEst       VALUES ('Veneto','Friuli')
                              TABLESPACE Ts2)
(PARTITION Vendite_G&F VALUES LESS THAN ('01-MAR-2004'),
 PARTITION Vendite_M&A VALUES LESS THAN ('01-MAY-2004')))
```

Definizione attributi

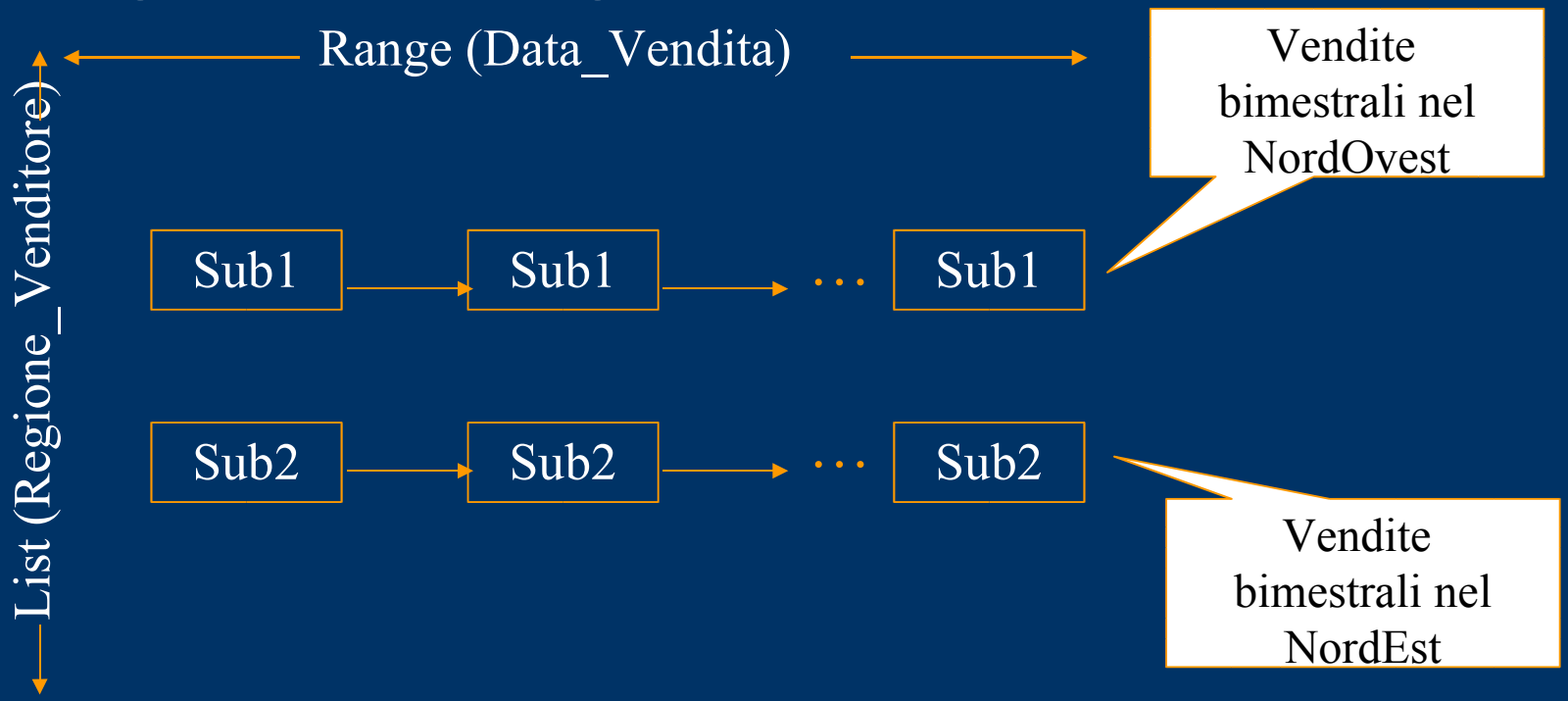
Definizione modalità utilizzate

Definizione sottopartizioni

Definizione partizioni

Composite Partitioning / 6 – esempio Range - List

- La tabella precedente può essere letta nel seguente modo grafico:



Partizionamento di Indici

- Tipi di indice
- Indici locali
- Indici globali
- Esempi

Tipi di indice

Unique/Nonunique

- La chiave utilizzata è unica
- Creazione di indice unique preferibile a vincolo unique nella tabella di riferimento

Composto

- E' riferito a un insieme di colonne
- Utile se le colonne vengono riferite insieme nelle istruzioni di select

Tipi di indice

Organizzazione

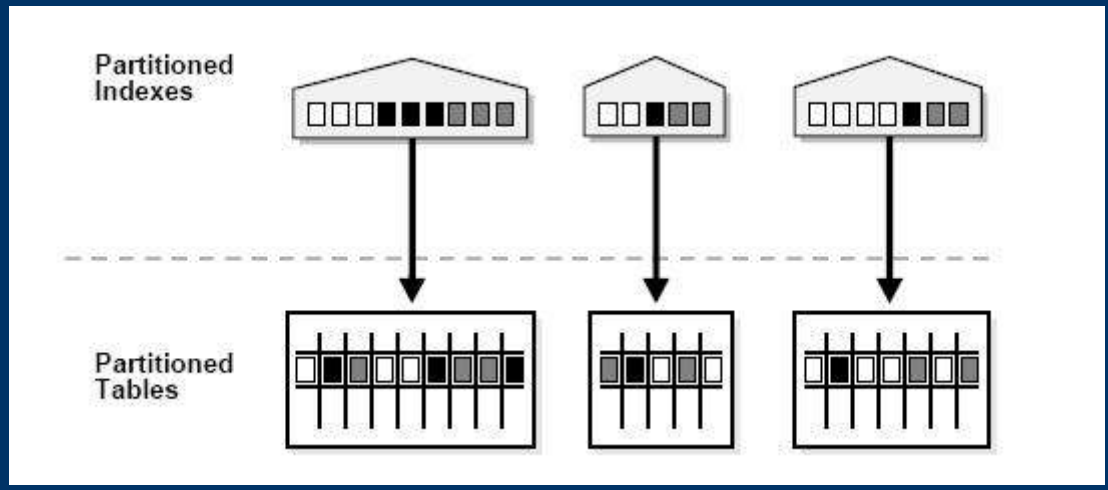
- B Tree
- Reverse Key
- Bitmap

Indici locali

- Facili da gestire
- Collegati in modo diretto alle tabelle
- Diffusi in ambienti DSS
- Adatti ad applicazioni OLTP

Indici locali - funzionamento

- Ogni partizione dell'indice è associata con una sola partizione della tabella
- Non è possibile aggiungere esplicitamente una partizione all'indice

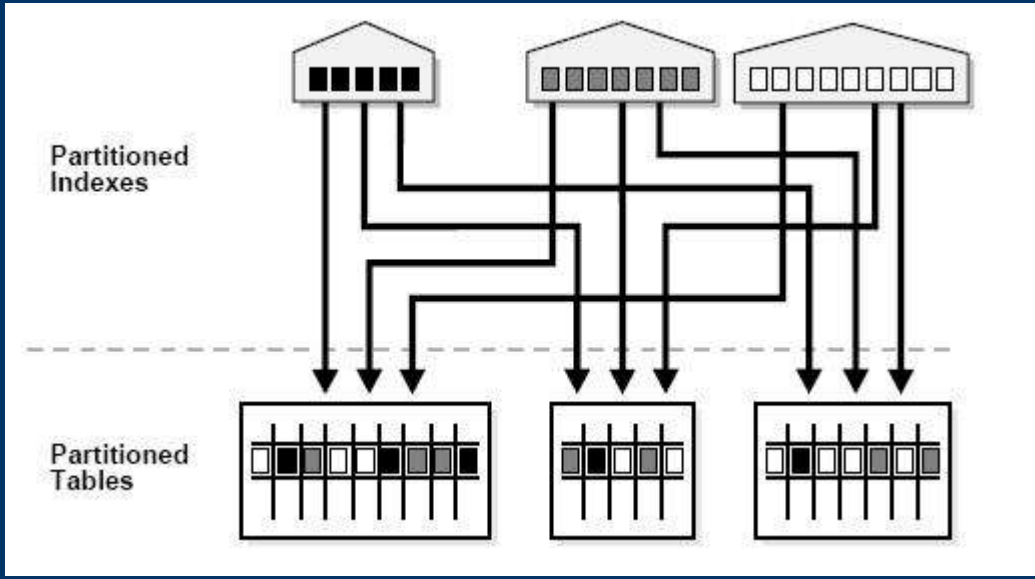


Indici locali - esempio

```
CREATE TABLE Impiegati
(id_impiegato NUMBER(4) NOT NULL,
cognome VARCHAR(15),
id_dipartimento NUMBER(2))
PARTITION BY RANGE (id_dipartimento)
(PARTITION impiegati_1 VALUES LESS THAN (10) TABLESPACE parte1,
PARTITION impiegati_2 VALUES LESS THAN (20) TABLESPACE parte2,
PARTITION impiegati_3 VALUES LESS THAN (30) TABLESPACE parte3);
CREATE INDEX indice_locale ON Impiegati (id_impiegato) LOCAL;
```

Indici globali

- Indipendenti dal partizionamento della tabella
- Diffusi in ambienti OLTP



Indici globali - operazioni

- Necessario definire un upper bound in fase di creazione
- Aggiunta di una partizione richiede split
- Necessità di usare la clausa UPDATE GLOBAL INDEXES per ogni modifica delle partizioni della tabella
- Non è necessario ricostruire l'indice in seguito a una modifica della partizione

Indici globali - esempio

```
CREATE INDEX indice_globale ON Impiegati (id_impiegato)
GLOBAL PARTITION BY RANGE (id_impiegato)
( PARTITION partizione1 VALUES LESS THAN (100),
  PARTITION partizione2 VALUES LESS THAN (MAXVALUE));

ALTER INDEX indice_globale SPLIT
PARTITION partizione1 AT VALUES LESS THAN (50) INTO
(PARTITION partizione1_a TABLESPACE t1,
PARTITION partizione1_b TABLESPACE t2);
```

Indici e OLTP

Globali

- Prestazioni maggiori con indici globali e unique rispetto a nonunique

Locali

- Aumento di availability in caso di manutenzioni frequenti delle partizioni
- Aumento delle prestazioni con controllo in parallelo rispetto alla chiave

Indici su partizioni composte

- Solo indici globali partizionati in base a range
- In caso di subpartizionamento gli indici sono locali e memorizzati assieme alla partizione della tabella di default
- Possibilità di indicare tablespace a livello di subpartizione

Partizionamento per migliorare le performance

- Partition Pruning
- Partition-wise Joins
- Parallel DML

Partition Pruning

- Elimina l'analisi delle partizioni non necessarie nella query
- Nel caso che una sola partizione soddisfi l'interrogazione, la clausola WHERE viene eliminata dallo statement SQL per migliorare le performance
- Il pruning può essere applicato anche agli indici partizionati sia locali che globali
- Riduce sensibilmente il volume dati da analizzare
- Elementi analizzati per il partition pruning:
  - Equality, range, LIKE, IN-list → range/list partition
  - Equality, IN-list → hash partition

Partition Pruning - Esempio

```
CREATE TABLE orders
(value NUMBER(10),
order_date DATE)
PARTITION BY RANGE(order_date)
(
PARTITION order_jan2005 VALUES LESS THAN(TO_DATE('01/02/2005','DD/MM/YYYY')),
PARTITION order_feb2005 VALUES LESS THAN(TO_DATE('01/03/2005','DD/MM/YYYY')),
PARTITION order_mar2005 VALUES LESS THAN(TO_DATE('01/04/2005','DD/MM/YYYY')),
PARTITION order_apr2005 VALUES LESS THAN(TO_DATE('01/05/2005','DD/MM/YYYY')),
PARTITION order_may2005 VALUES LESS THAN(TO_DATE('01/06/2005','DD/MM/YYYY')),
PARTITION order_jun2005 VALUES LESS THAN(TO_DATE('01/07/2005','DD/MM/YYYY'))
);

SELECT SUM(value)
FROM orders
WHERE order_date BETWEEN '28-MAR-98' AND '23-APR-98‘;
```

18. Si eliminano le partizioni:

    order_jan2005, order_feb2005, order_may2005, order_jun2005

19. Si fa un **full scan** o un **index scan** sulle partizioni rimanenti:

    order_mar2005, order_apr2005

Partition-Wise Joins

- Utilizzato per eseguire delle join su due tabelle che sono entrambe (o solamente una) partizionate sulle colonne di join.
- Due tipologie:
  - Full
  - Partial (più comune)
- Vantaggi
  - Riduzione di overhead nelle comunicazioni (in Oracle Real Application Clusters)
  - Maggiore scalabilità
  - Riduzione requisiti di memoria
- Cost-based optimizer valuta se usare o meno il partition-wise join:
  - Dimensioni delle partizioni simili
  - Numero di partizioni multipla rispetto ai server
  - I/O remoto supplementare

Full Partition-Wise Joins

- Divide una grossa join in join più piccole eseguite tra le partizioni delle tabelle in oggetto
- Metodi di join:
  - **Hash-Hash**
  - **(Composite-Hash)-Hash**
  - **(Composite-List)-List**
  - **Composite-Composite (Hash/List Dimension)**
  - **Range-Range e List-List**
  - **Range-Composite, Composite-Composite (Range Dimension)**

Partition-Wise Joins – Esempio

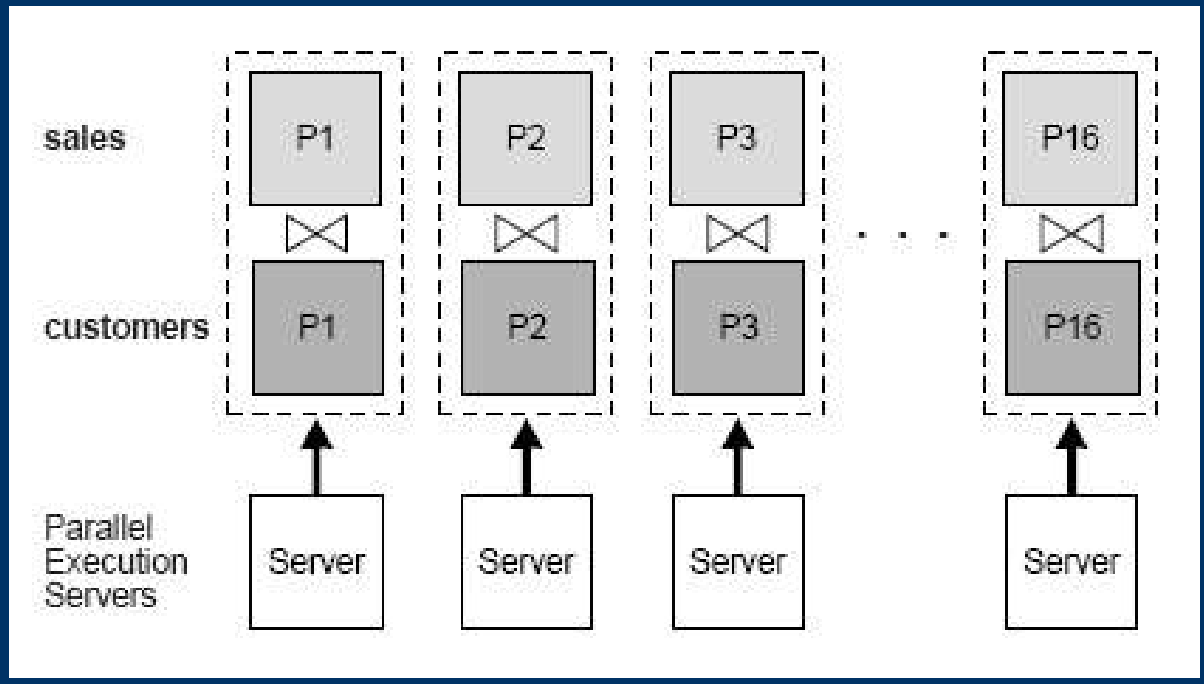
```
CREATE TABLE sales
(productid NUMBER(5),
date DATE,
cust_id NUMBER(5),
totalprice NUMBER(4) );

CREATE TABLE customers
(id NUMBER(5),
name VARCHAR2(30),
article VARCHAR2(1000) );

SELECT c.cust_last_name, COUNT(*)
FROM sales s, customers c
WHERE s.cust_id = c.cust_id AND
     s.date BETWEEN TO_DATE('01-JUL-1999', 'DD-MON-YYYY') AND
     (TO_DATE('01-OCT-1999', 'DD-MON-YYYY'))
GROUP BY c.name HAVING COUNT(*) > 100;
```

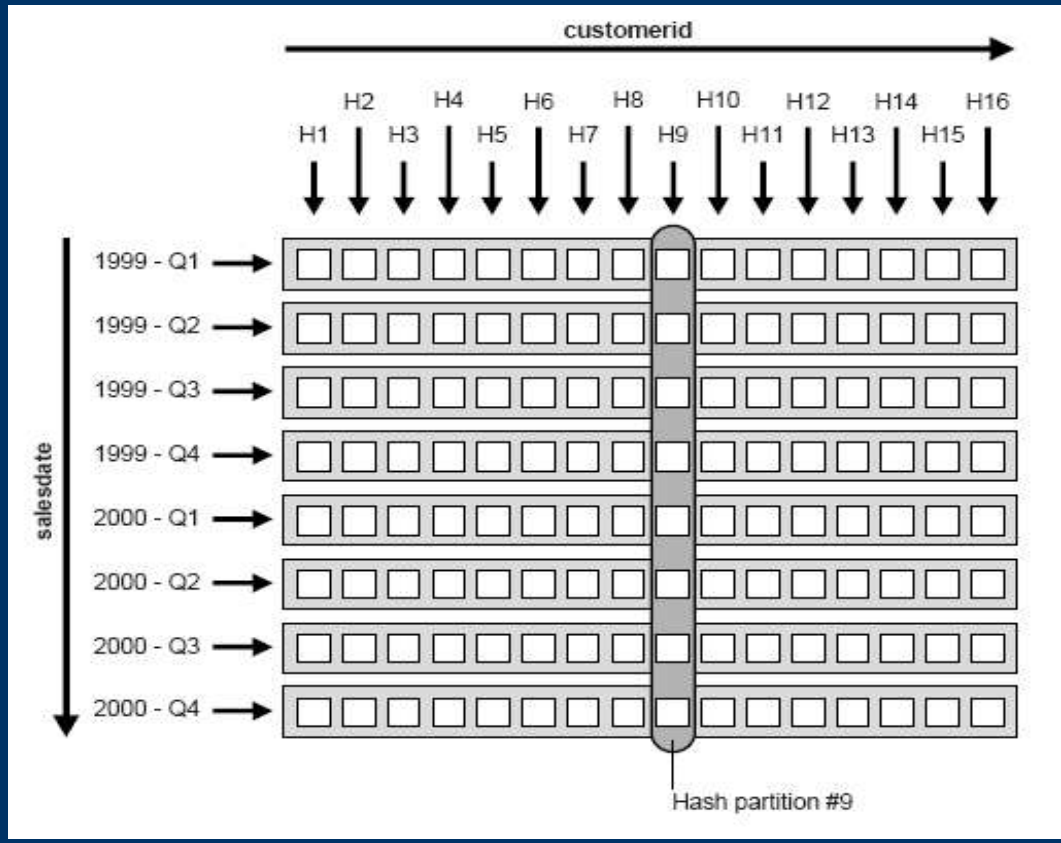
Hash-Hash

- Metodo più semplice (il sistema decide come suddividere equamente i dati nelle partizioni indicate)
- Massimo 16 partizioni



(Composite-Hash)-Hash

- Massimo 16 partizioni per hash
- Ogni partizione hash dev'essere sullo stesso nodo
- Stesso concetto per:
  - (Composite-List)-List
  - Composite-Composite



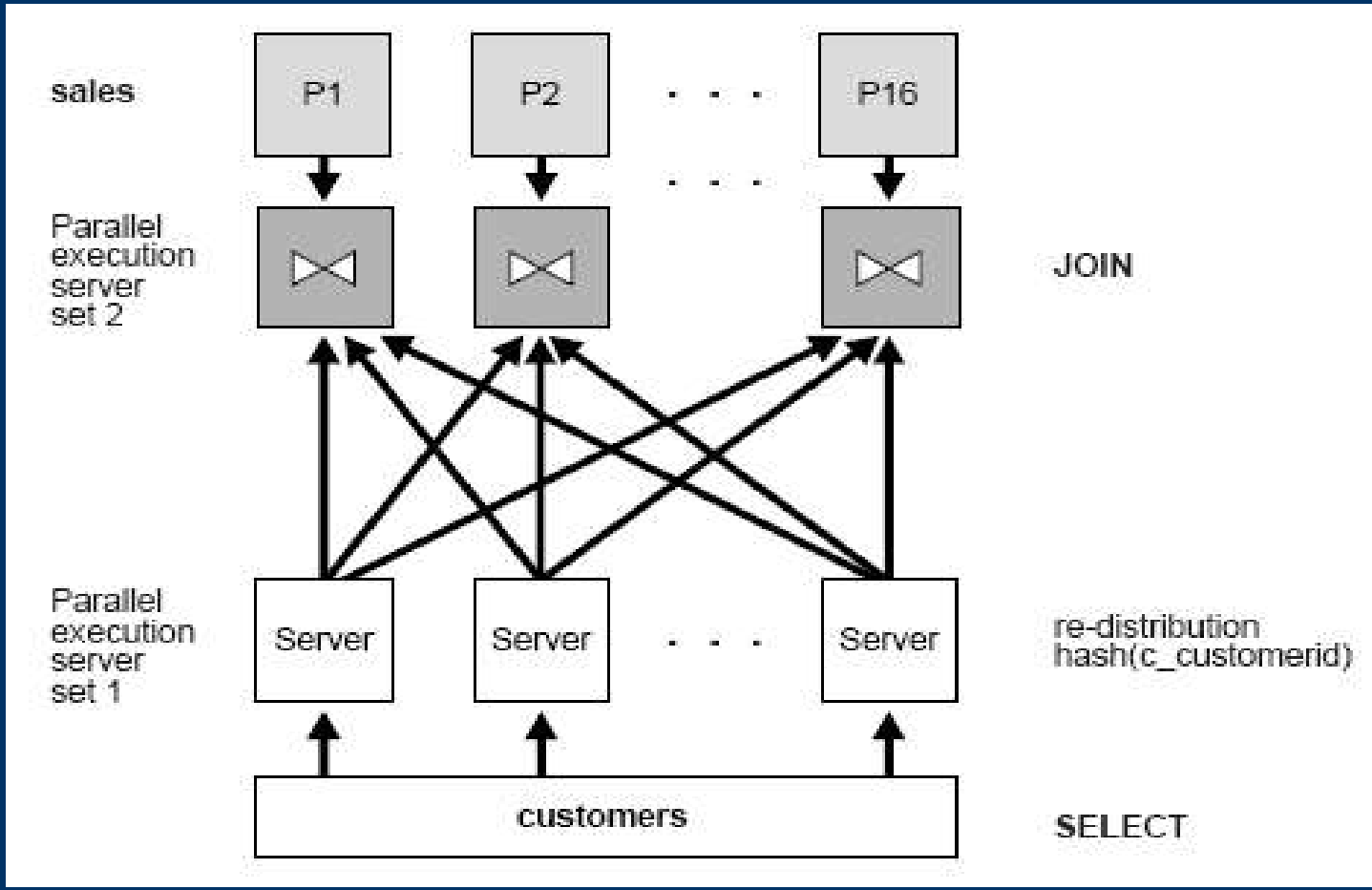
Range-Range e List-List

- Poco utilizzato
- Gestione manuale delle partizioni
- Conoscenza a priori del carico di dati che si avrà sulle varie partizioni (bilanciamento manuale del carico)
- Si può anche fare il partition-wise join sulla partizione range del composite
  - Range-Composite
  - Composite-Composite (Range Dimension)

Partial Partition-Wise Joins(1)

- Basta una sola tabella partizionata (reference table)
- La tabella non partizionata viene ridistribuita nei vari server in base alla partizione di reference table presente sul server
- Scambio di righe tra server! (usare Pk o Fk)
- 2 livelli di server:
  1. Ridistribuisce la tabella non partizionata ai server di livello 2
  2. Esegue la join delle righe fornite con la partizione di reference table in possesso
- Metodi:
  - Hash/List
  - Composite
  - Range

Partial Partition-Wise Joins(2)



Parallel DML(1)

- Usato per velocizzare o rendere scalabile DML su tabelle o indici di grosse dimensioni
- Parallel DML manualmente:
  - Eseguire INSERT multipli a diverse istanze di una Oracle Real Application Clusters per usare lo spazio libero su diversi blocchi free list.
  - Eseguire UPDATE e DELETE multipli con rowid range o range di valori di chiave differenti.
- Eseguire DML parallele manualmente ha i seguenti svantaggi:
  - E' difficile da usare!
  - Bisogna coordinare manualmente i commit o i rollback dei vari statement in modo da mantenere l'atomicità delle operazioni che si vuole svolgere parallelamente.
  - Il calcolo per la divisione dei compiti può essere comlpesso.

Parallel DML(2)

- Oracle può eseguire parallel DML automaticamente
- Comandi:
  - ALTER SESSION ENABLE PARALLEL DML
  - ALTER SESSION DISABLE PARALLEL DML (default)
- Obbligatorio perché le DML parallele hanno diversi requisiti di locking, transaction, disk space.
- Attenzione! Non tutto è parallelizzabile!
- In una transazione una tabella può essere acceduta da una sola statement parallela!
- Oracle supporta recovery su parallel DML (parallel rollback)