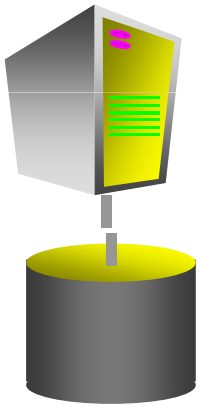
Algumas opções do SGBDR Oracle

índice

- **Arquitecturas multi-instâncias**
  - ***Single instance versus Two instances architecture***
  - ***Real Application Cluster***
  - ***Shared-Database vs. Shared-Nothing Clusters***
  - ***Cache Fusion***
- ***Table Partitioning***
  - ***Range***
  - ***Hash***
  - ***List***
  - ***Composite***

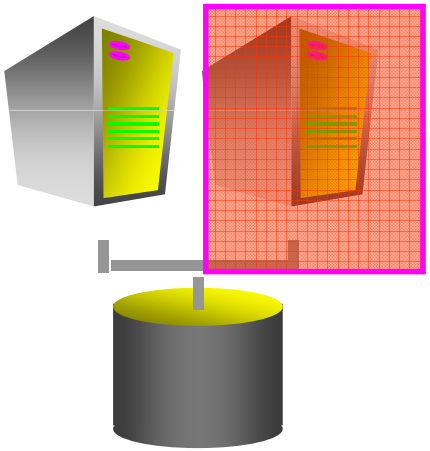
Arquitecturas *Single instance vs Multi instances*

1 instância & 1 BD



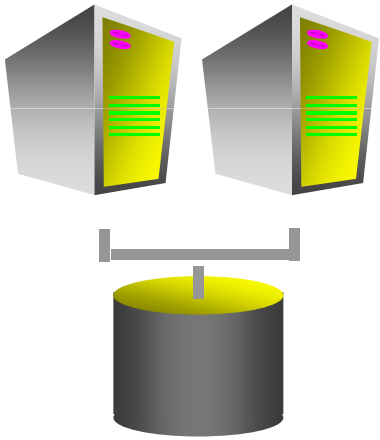
Default

1 instância & 1BD



Failover

2 instâncias & 1BD



OPS / RAC

Real Application Clusters - RAC

- **O RAC permite alta disponibilidade e escalabilidade, uma vez que permite ter vários nós activos em simultâneo sobre a mesma Base de Dados.**

- **Permite a distribuição da execução das aplicações sem modificações no código aplicacional, aumentando o aproveitamento dos recursos.**

- **A opção *Transparent Application Failover* serve de complemento ao nível de disponibilidade da BD, pois permite que os utilizadores sejam reconectados de forma transparente a outro nó.**
  - **As *queries* continuam a execução no ponto em que foram interrompidas.**

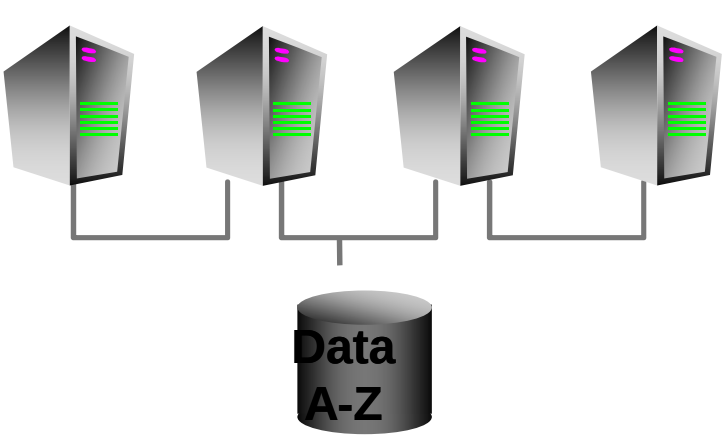
Shared-Database vs. Shared-Nothing

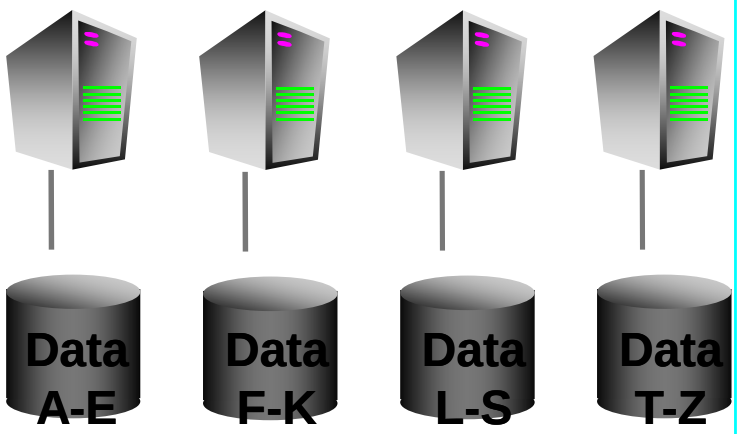
Shared Disk

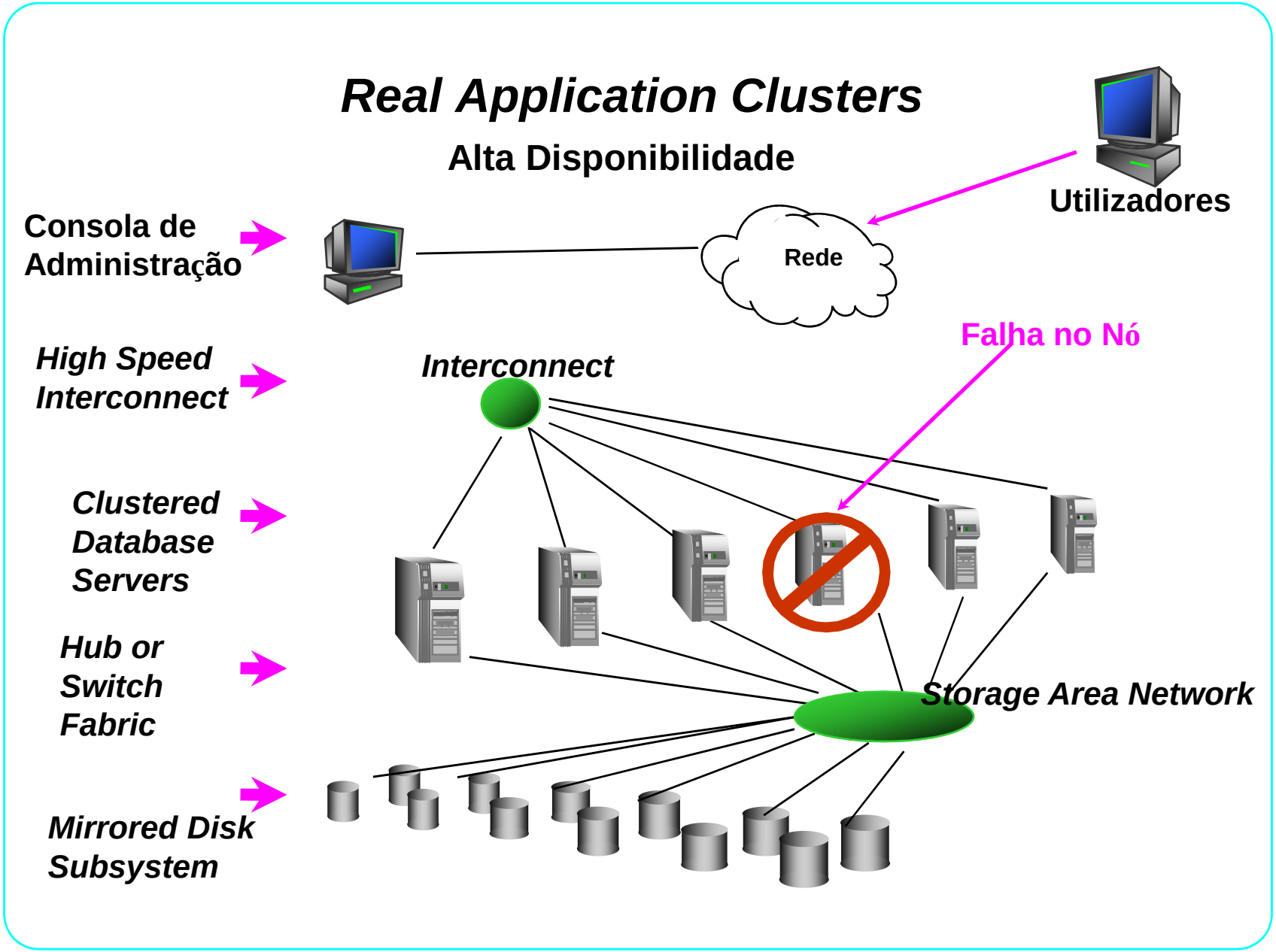
- **Oracle e *Mainframes* IBM**
- **Mais fiável à medida que adicionamos mais nós.**
- **Não necessita do particionamento dos dados.**

Shared Nothing

- **Microsoft e IBM Unix/NT**
- **Menor Fiabilidade**
- **Particionamento dos dados**



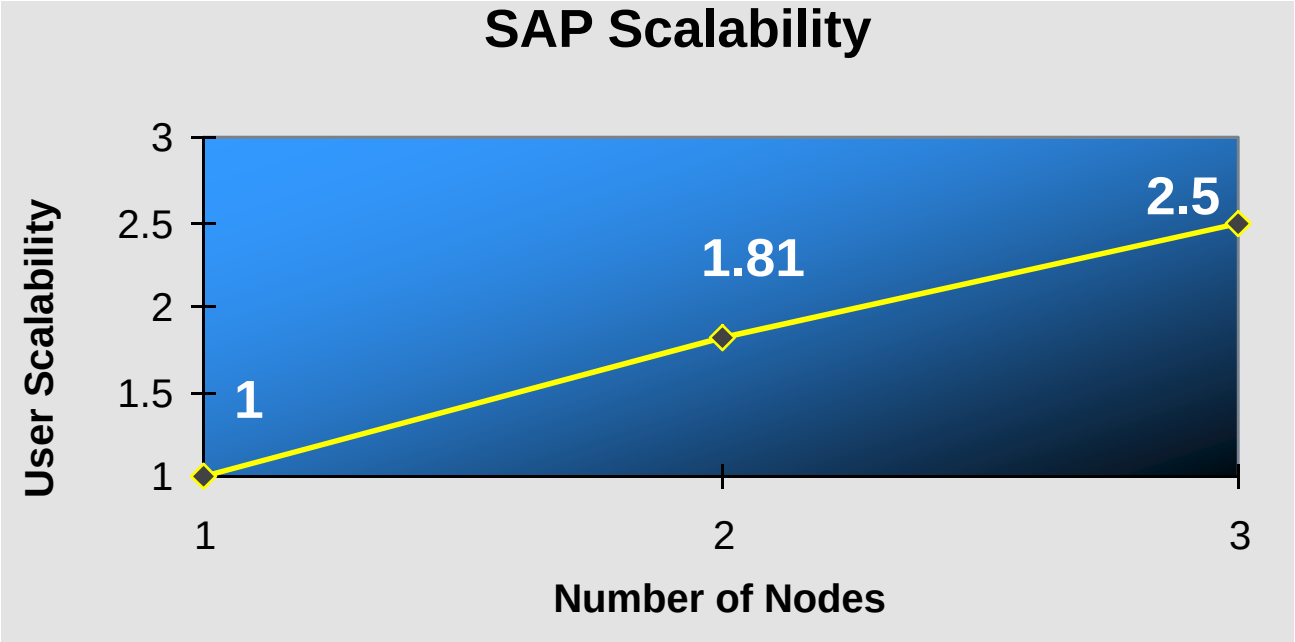




Real Application Clusters

Escalabilidade

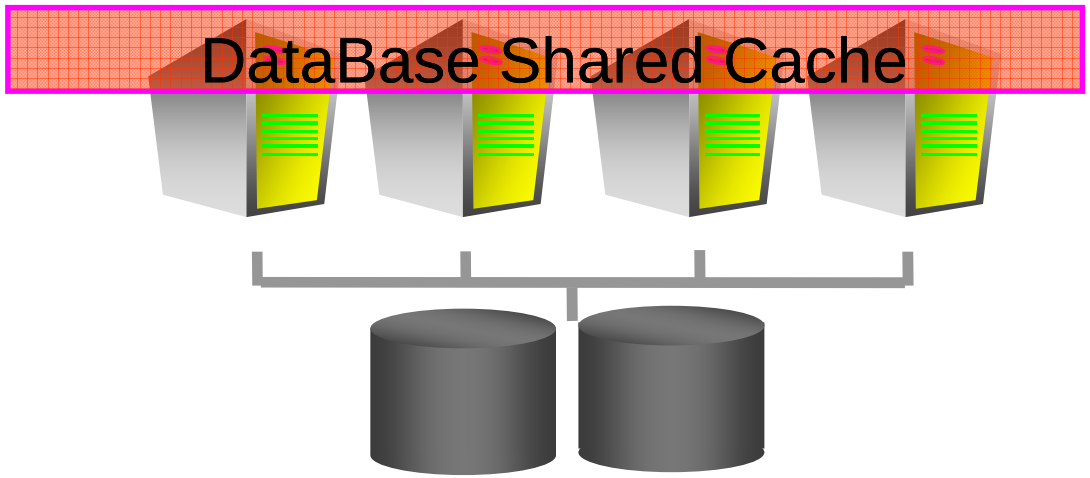
Adicionando mais nós, aumentamos a capacidade do sistema.



Real Application Clusters

Performance

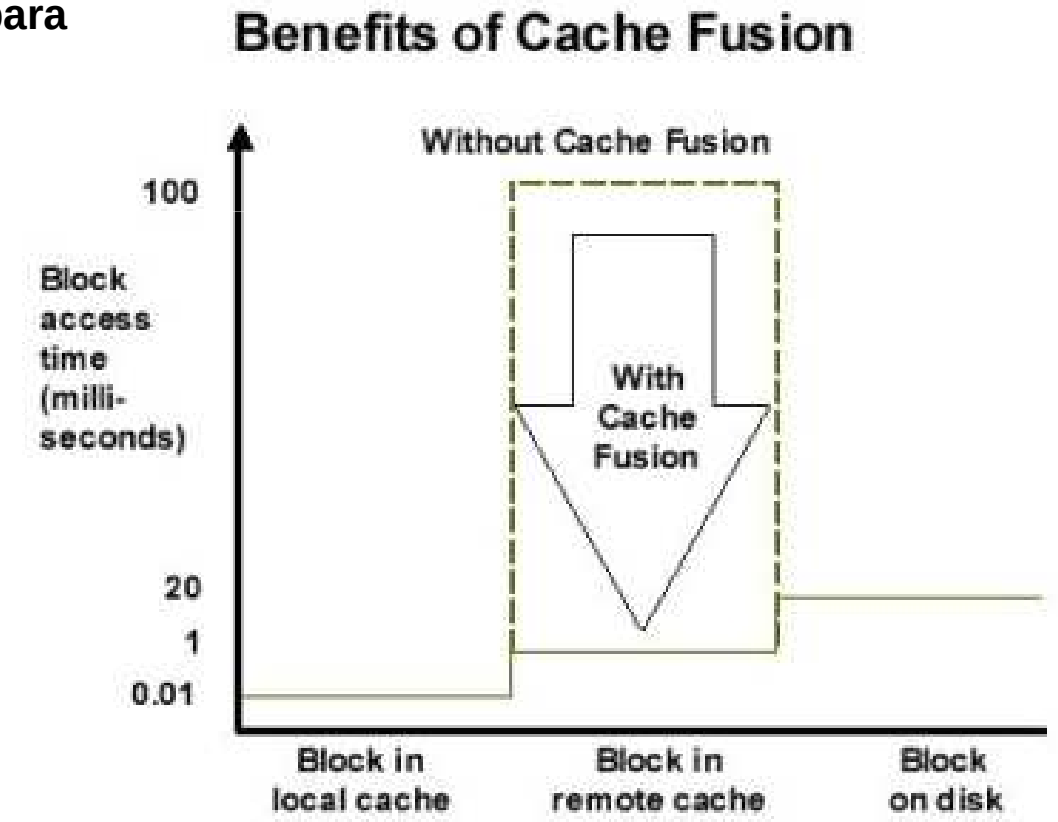
Esta arquitectura permite ultrapassar as limitações das aproximações *shared nothing* e *shared disk* tornando os sistemas escaláveis, com maior desempenho e maior disponibilidade.



Real Application Clusters

Performance

Explora a tecnologia emergente dos *inteconnects*, retirando as operações lentas realizadas em disco para memória.



Particionamento

Particionamento

Endereça o problema de suportar tabelas e índices muito grandes permitindo a sua decomposição em pedaços mais pequenos e de melhor desempenho (partições).

Uma vez definidas e criadas, as instruções SQL podem aceder e manipular os dados directamente nas partições, evitando o uso da tabela completa.

Utilizam-se sobretudo em aplicações de *Data Warehouse* onde se trabalham grandes quantidades de informação de histórico.

Particionamento simples

- ***Range partitioning***

  Os dados da tabela ou índice são agrupados de acordo com uma escala de valores.

- ***Hash partitioning***

  Usa uma função de *hash* para colocar os dados nas partições permitindo que todos os dados tirem partido dos benefícios do particionamento.

- ***List partitioning***

  Permite o controlo explícito relativamente ao mapeamento das linhas.

  Especifica uma lista de valores distintos para a coluna particionada.

  Suporta uma partição *default*. A chave da partição é composta por uma só coluna da tabela ao invés dos restantes métodos.

Particionamento simples

- ***System partitioning***

  Controlo aplicacional do particionamento.

- ***Reference partitioning***

  Particionamento por *referencial constraint*.

- ***Interval partitioning***

  Uma extensão do *range*. Novas partições são criadas automáticamente.

Particionamento composto

- ***Range-Hash partitioning***
  Optimização na distribuição dos dados por várias sub-partições.

- ***Range-List partitioning***
  Vantagem de administração do *range* com o controlo explicito do *list partitioning*.

- ***Range-range partitioning***
  Range* em 2 dimensões. Por exemplo, particionar por *order_date* e subparticionar por *shipping_date.

- ***List-range partitioning***
  Possibilita a partição lógica dado uma determinada lista (partição). Por exemplo, particionar por *country_id* e subpartição *order_date.*

- ***List-hash partitioning***
  Dados particionados por *list* e subparticionados por *hash*. P*artition-wise joins.*

- ***List-list partitioning***
  Partição de c*ountry_id* e subpartição por *sales_channel.*

Particionamento

- ***Range Partitioning***

```
CREATE TABLE sales_range
(salesman_id  NUMBER(5),
salesman_name VARCHAR2(30),
sales_amount  NUMBER(10),
sales_date    DATE)
PARTITION BY RANGE(sales_date)
(
PARTITION sales_jan2000 VALUES LESS THAN(TO_DATE('02/01/2000','DD/MM/YYYY')),
PARTITION sales_feb2000 VALUES LESS THAN(TO_DATE('03/01/2000','DD/MM/YYYY')),
PARTITION sales_mar2000 VALUES LESS THAN(TO_DATE('04/01/2000','DD/MM/YYYY')),
PARTITION sales_apr2000 VALUES LESS THAN(TO_DATE('05/01/2000','DD/MM/YYYY'))
);
```

Particionamento

- ***Hash Partitioning***

```
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);
```

Particionamento

- ***List Partitioning***

```
CREATE TABLE sales_by_region
(deptno       number(5),
 deptname     VARCHAR2(30),
 quarterly_sales number(10,2),
 state        varchar2(2)
)
partition by list (state)
( partition q1_northwest       values ('OR','WA'),
  partition q1_southwest       values ('AZ','UT','NM'),
  partition q1_northeast       values ('NY','VM','NJ'),
  partition q1_southeast       values ('FL','GA'),
  partition q1_northcentral    values ('SD','WI'),
  partition q1_southcentral    values ('NT','TX'),
  partition q1_other           values (default)
);
```

Particionamento

- ***Interval Partitioning***

```
create table test
          (sno number(6),
          last_name varchar2(30),
          salary number(6))
          partition by range(salary)
          Interval  (5000)
         (
      partition p1 values less than (5000),
      partition p2 values less than (10000),
     partition p3 values less than (15000),
     partition p4 values less than (20000));
```

Particionamento

- ***Reference Partitioning***

```
CREATE TABLE order_items (
  order_id    NUMBER NOT NULL,
  product_id  NUMBER NOT NULL,
  price       NUMBER,
  quantity    NUMBER,
  CONSTRAINT order_items_fk FOREIGN KEY (order_id) REFERENCES
   orders)
PARTITION BY REFERENCE (order_items_fk);
```

Particionamento

- ***System Partitioning***

```
CREATE TABLE syspart
(c1 NUMBER,
 c2 NUMBER)
 PARTITION BY SYSTEM
(PARTITION p1 TABLESPACE ts1,
 PARTITION p2 TABLESPACE ts2,
 PARTITION p3 TABLESPACE ts3 );

INSERT INTO syspart PARTITION p1 VALUES (1, 2);
INSERT INTO syspart PARTITION p2 VALUES (3, 4);
```

Particionamento

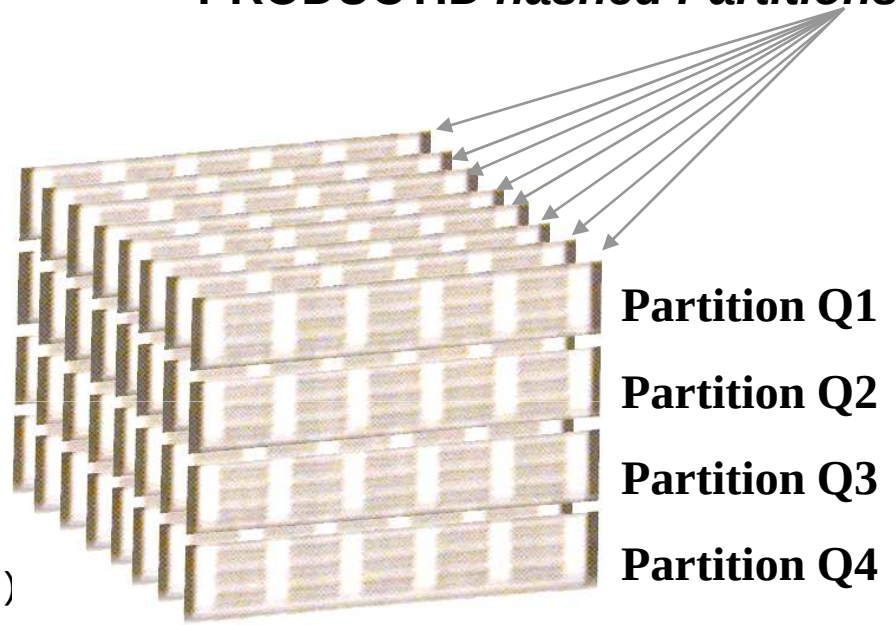
- ***Composite range/hash***

  Dentro de cada partição por *range*, os dados são *hashed* em subpartições.

- ***Composite range/list***

  Adiciona à facilidade de descrição do *range partition* o controlo explicito herdado do *list partitioning*.

Particionamento

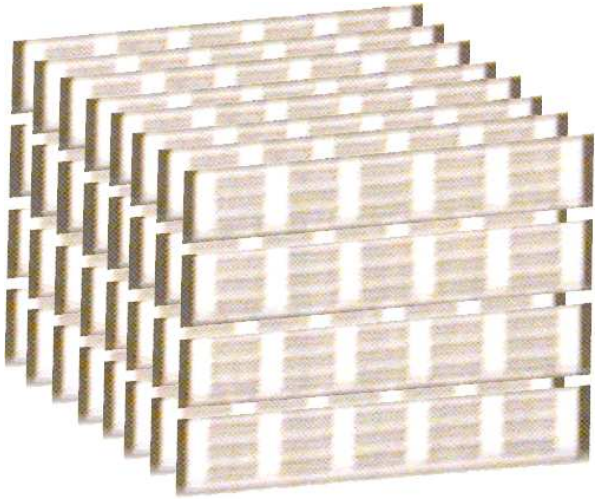


- ***Range-Hash***

```
CREATE TABLE orders (
ordid  NUMBER,
orderdate DATE,
productid NUMBER,
quantity NUMBER)
PARTITION BY RANGE (ordid)
SUBPARTITION BY HASH (productid)
SUBPARTITIONS 8 STORE IN (ts1, ts2, ts3,ts4,ts5,ts6,ts7,ts8)
(
PARTITION q1 VALUES LESS THAN (1000),
PARTITION q2 VALUES LESS THAN (2000),
PARTITION q3 VALUES LESS THAN (3000),
PARTITION q4 VALUES LESS THAN (MAXVALUE));
```

- ***Range list***

PRODUCTID listados em subpartições



Partition Q1

Partition Q2

Partition Q3

Partition Q4

```
CREATE TABLE orders (
ordid  NUMBER,
orderdate DATE,
productid VARCHAR2(2),
quantity NUMBER)
PARTITION BY RANGE (ordid)
SUBPARTITION BY LIST (productid)
SUBPARTITION TEMPLATE
(
SUBPARTITION prodA values ('AA','AB','AC','AD') TABLESPACE ts1,
SUBPARTITION prodB values ('BA','BB','BC','BD') TABLESPACE ts2,
SUBPARTITION prodC values ('CA','CB','CC','CD') TABLESPACE ts3,
SUBPARTITION prodD values ('DA','DB','DC','DD') TABLESPACE ts4,
SUBPARTITION prodE values ('EA','EB','EC','ED') TABLESPACE ts5,
SUBPARTITION prodF values ('FA','FB','FC','FD') TABLESPACE ts6,
SUBPARTITION prodG values ('GA','GB','GC','GD') TABLESPACE ts7,
SUBPARTITION prodH values ('HA','HB','HC','HD') TABLESPACE ts8)
(
PARTITION q1 VALUES LESS THAN (1000),
PARTITION q2 VALUES LESS THAN (2000),
PARTITION q3 VALUES LESS THAN (3000),
PARTITION q4 VALUES LESS THAN (MAXVALUE));
```

Particionamento

- ***Partition Pruning***

  O Oracle *server* consegue reconhecer as partições e subpartições e usar este conhecimento para a optimização do SQL "eliminando" (*pruning*) partições desnecessárias.

- **Por exemplo, se a *query* envolve apenas os dados de Q1, não é necessário ir buscar os dados dos *Quarters* restantes. Assim, reduz-se dramaticamente o volume de dados, resultando em melhorias substanciais no desempenho da consulta.**

Particionamento

- **Vantagens**

  As seguintes classes apresentam-se como candidatas aos benefícios do particionamento:

- ***Very Large Databases (VLDBs)***
- ***DSS Performance***
- ***I/O Performance***
- ***Partition Transparency***
- ***ILM (Information Lifecycle Management)*** **–eficiência no custo do *storage.***