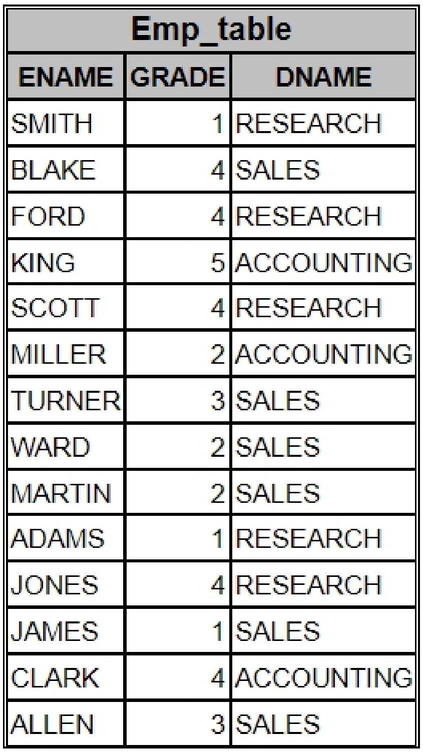
Problem Solving on Round Robin, Hash and Range Partitioning

Let us start with the following table **Emp_table**. Emp_table instance has 14 records and every record stores information about the name of the employee; his/her work grade, and the department name. Assume that we have 3 processors namely P_0, P_1, P_2, and 3 Disks associated with those 3 processors namely D_0, D_1, D_2.

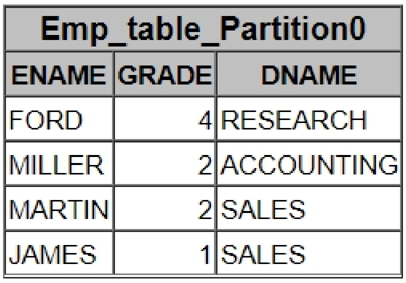


Emp_table		
ENAME	**GRADE**	**DNAME**
SMITH	1	RESEARCH
BLAKE	4	SALES
FORD	4	RESEARCH
KING	5	ACCOUNTING
SCOTT	4	RESEARCH
MILLER	2	ACCOUNTING
TURNER	3	SALES
WARD	2	SALES
MARTIN	2	SALES
ADAMS	1	RESEARCH
JONES	4	RESEARCH
JAMES	1	SALES
CLARK	4	ACCOUNTING
ALLEN	3	SALES

Table 1 – Emp_table

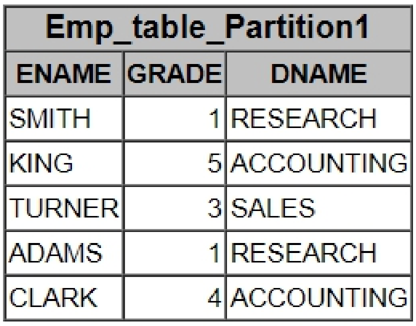
A. Round-Robin Partitioning:

In this strategy we partition records in a round-robin manner using the function **i mod n**, where i is the record position in the table and n is the number of partitions/disks which is in our case 3. On the application of partitioning technique, first record goes into D1, second record goes into D2, third record goes into D0, fourth record goes into D1, and so on. After distribution of records, we will get the following partitions;



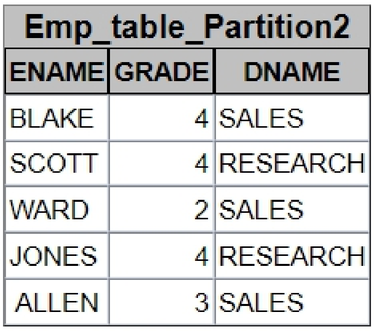
Emp_table_Partition0		
ENAME	GRADE	DNAME
FORD	4	RESEARCH
MILLER	2	ACCOUNTING
MARTIN	2	SALES
JAMES	1	SALES

Table 2 – Records 3, 6, 9, 12 mod 3



Emp_table_Partition1		
ENAME	GRADE	DNAME
SMITH	1	RESEARCH
KING	5	ACCOUNTING
TURNER	3	SALES
ADAMS	1	RESEARCH
CLARK	4	ACCOUNTING

Table 3 – Records 1, 4, 7, 10, 13 mod 3



Emp_table_Partition2		
ENAME	GRADE	DNAME
BLAKE	4	SALES
SCOTT	4	RESEARCH
WARD	2	SALES
JONES	4	RESEARCH
ALLEN	3	SALES

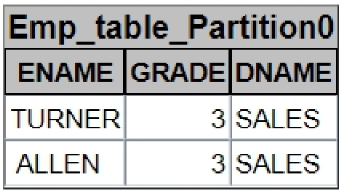
Table 4 – Records 2, 5, 8, 11, 14 mod 3

B. Hash Partitioning:

Let us take *GRADE* attribute of the Emp_table to explain Hash partitioning. Let us choose a hash function as follows;

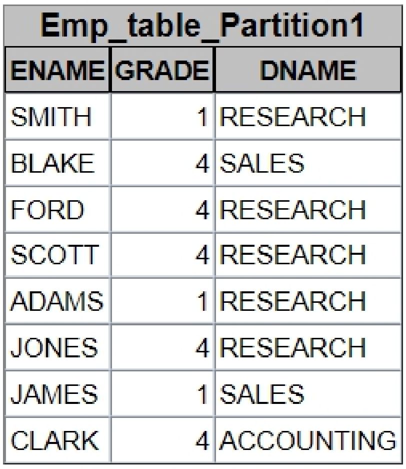
h(GRADE) = (GRADE mod n)

where GRADE is the value of *GRADE* attribute of a record and n is number of partitions which is 3 in our case. While applying the hash partitioning on GRADE, we will get the following partitions of Emp_table. For example, the GRADE of 'Smith' is 1 and while hashing the function shows partition 1 (i.e 1 mod 3 = 1). The GRADE of 'Blake' is 4, then (4 mod 3) directs to partition 1. The GRADE of 'King' is 5 which directs to partition 2 (5 mod 3 = 2).



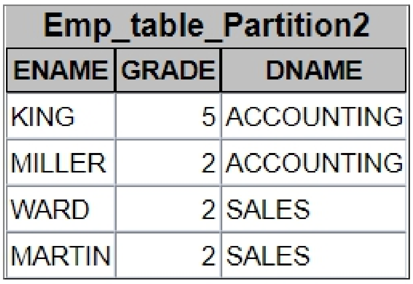
Emp_table_Partition0		
ENAME	GRADE	DNAME
TURNER	3	SALES
ALLEN	3	SALES

Table 5 – GRADEs 3 mod 3



Emp_table_Partition1		
ENAME	GRADE	DNAME
SMITH	1	RESEARCH
BLAKE	4	SALES
FORD	4	RESEARCH
SCOTT	4	RESEARCH
ADAMS	1	RESEARCH
JONES	4	RESEARCH
JAMES	1	SALES
CLARK	4	ACCOUNTING

Table 6 – GRADEs 1, 4 mod 3



Emp_table_Partition2		
ENAME	GRADE	DNAME
KING	5	ACCOUNTING
MILLER	2	ACCOUNTING
WARD	2	SALES
MARTIN	2	SALES

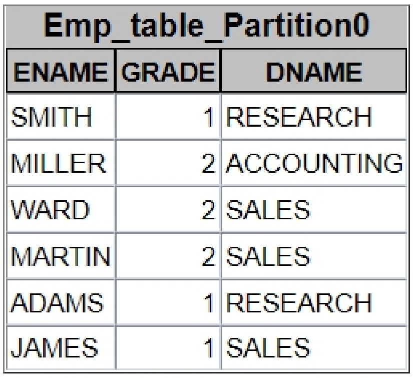
Table 7 – GRADEs 2, 5 mod 3

C. Range Partitioning:

Let us consider *GRADE* of Emp_table to partition under range partitioning. For applying range partition, we need to first identify partitioning vector, [v0, v1, …, vn-2]. Let us choose the following vector as range partitioning vector for our case;

[2, 4]

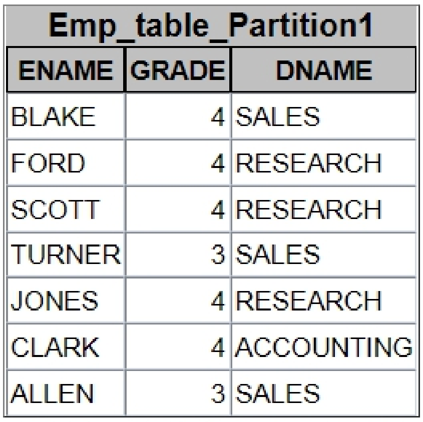
According to the vector, the records having the *GRADE* value 2 and less will go into partition 0, greater than 2 and less than or equal to 4 will go into partition 1, and all the other values (greater than 4) will go into partition 2 as depicted in the following tables.



Emp_table_Partition0

ENAME	GRADE	DNAME
SMITH	1	RESEARCH
MILLER	2	ACCOUNTING
WARD	2	SALES
MARTIN	2	SALES
ADAMS	1	RESEARCH
JAMES	1	SALES

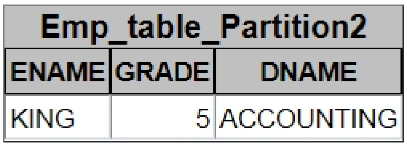
Table 8 – GRADE values 1 and 2



Emp_table_Partition1

ENAME	GRADE	DNAME
BLAKE	4	SALES
FORD	4	RESEARCH
SCOTT	4	RESEARCH
TURNER	3	SALES
JONES	4	RESEARCH
CLARK	4	ACCOUNTING
ALLEN	3	SALES

Table 9 – GRADE values 3 and 4



Emp_table_Partition2

ENAME	GRADE	DNAME
KING	5	ACCOUNTING

Table 10 – GRADE value 5 and above