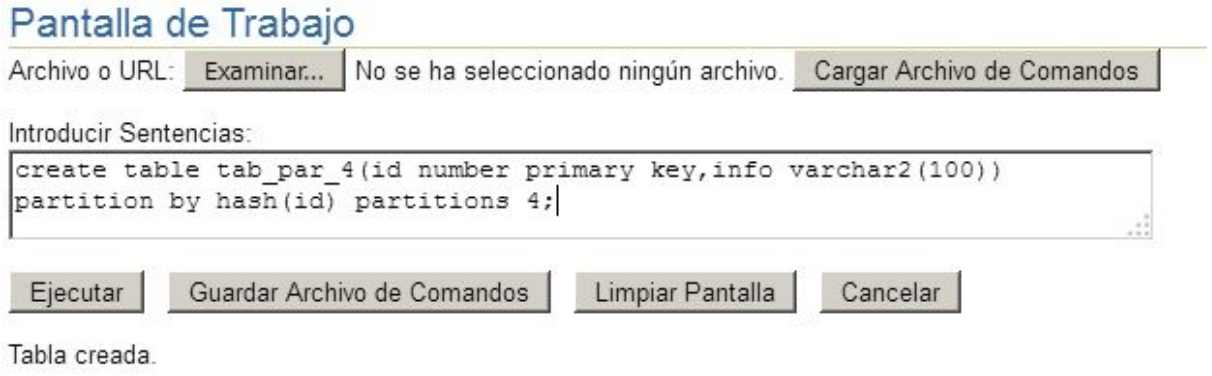
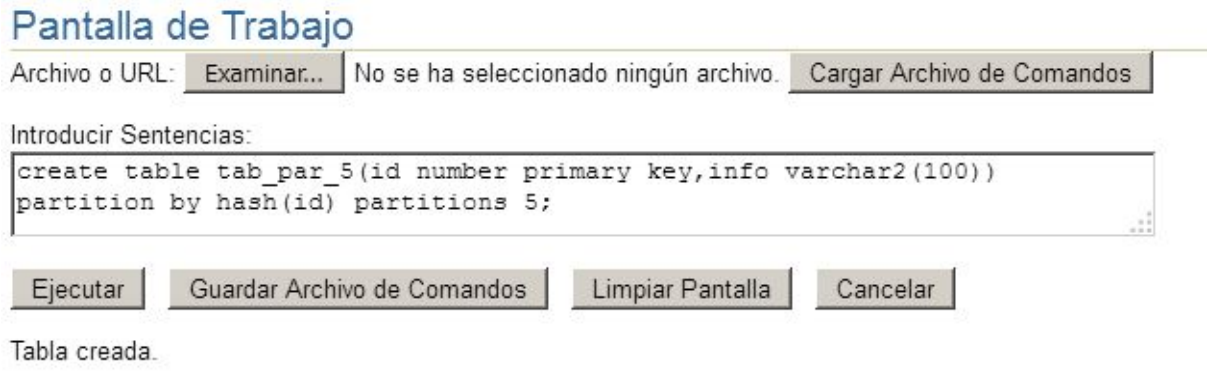
Ejercicio Partición:
Vamos a crear dos tablas con 10.000 ejemplos cada una, a la primera vamos a hacerle 4 particiones y la segunda 5 particiones.

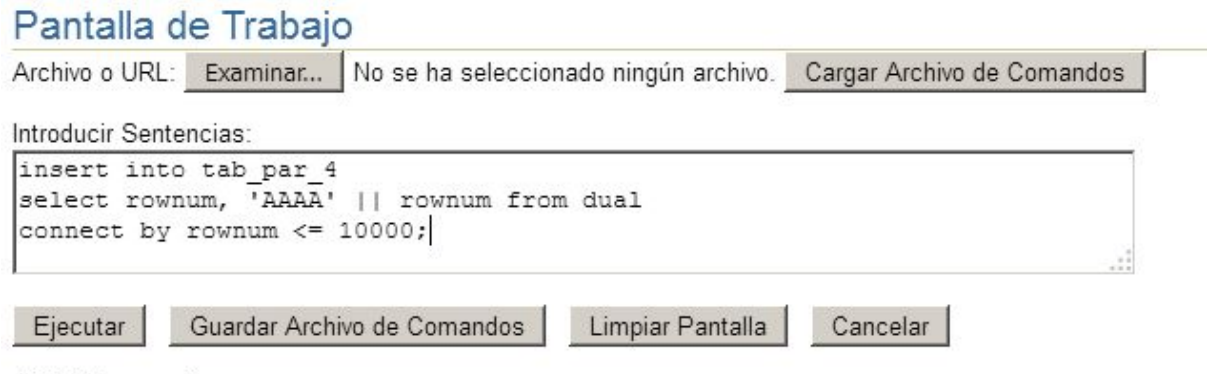
create table tab_par_4(id number primary key, info varchar2(100))
partition by hash(id) partitions 4;



create table tab_par_5(id number primary key, info varchar2(100))
partition by hash(id) partitions 5;



Insertamos 10000 valores en cada tabla:
insert into tab_par_4
select rownum, 'AAAA' || rownum from dual
connect by rownum <=100000;



insert into tab_par_5
select rownum, 'AAAA' || rownum from dual
connect by rownum <=100000;

Pantalla de Trabajo

Archivo o URL: Examinar... No se ha seleccionado ningún archivo. Cargar Archivo de Comandos

Introducir Sentencias:

```
insert into tab_par_5
select rownum, 'AAAA' || rownum from dual
connect by rownum <= 10000;
```

Ejecutar | Guardar Archivo de Comandos | Limpiar Pantalla | Cancelar

10000 filas creadas.

Comprobamos las particiones hechas:
select table_name, partition_name, num_rows, blocks
from dba_tab_partitions
where table_name in('TAB_PAR_4', 'TAB_PAR_5')
order by 1,2;

Pantalla de Trabajo

Archivo o URL: Examinar... No se ha seleccionado ningún archivo. Cargar Archivo de Comandos

Introducir Sentencias:

```
select table_name, partition_name, num_rows, blocks
from dba_tab_partitions
where table_name in ('TAB_PAR_4','TAB_PAR_5')
order by 1,2;
```

Ejecutar | Guardar Archivo de Comandos | Limpiar Pantalla | Cancelar

TABLE_NAME	PARTITION_NAME	NUM_ROWS	BLOCKS
TAB_PAR_4	SYS_P13		
TAB_PAR_4	SYS_P14		
TAB_PAR_4	SYS_P15		
TAB_PAR_4	SYS_P16		
TAB_PAR_5	SYS_P17		
TAB_PAR_5	SYS_P18		
TAB_PAR_5	SYS_P19		
TAB_PAR_5	SYS_P20		
TAB_PAR_5	SYS_P21		

9 filas seleccionadas.

Execution Plan

```
----------------------------------------------------------
 0    SELECT STATEMENT Optimizer=CHOOSE
 1  0   SORT (ORDER BY)
 2  1     VIEW OF 'DBA_TAB_PARTITIONS'
 3  2       UNION-ALL
 4  3         NEST ED LOOPS
 5  4         &nbs p;   NESTED LOOPS
 6  5         &nbs p;     NESTED LOOPS
 7  6         &nbs p;       NESTED LOOPS
 8  7         &nbs p;         TABLE AC CESS (FULL) OF 'TABPART$'
 9  7         &nbs p;         TABLE AC CESS (BY INDEX ROWID) OF 'OBJ$'
10  9         &nbs p;          & nbsp; INDEX (UNIQUE SCAN) OF 'I_OBJ1' (UNIQUE)
11  6         &nbs p;       TABLE ACCESS (CLUSTER) OF 'USER$'
12 11         &nbs p;         INDEX (U NIQUE SCAN) OF 'I_USER#' (NON-UNIQUE)
13  5         &nbs p;     TABLE ACCESS (CLUSTER) OF 'SEG$'
14 13         &nbs p;       INDEX (UNIQUE SCAN) OF 'I_FILE#_BLOCK#' (NON-UNIQUE)
15  4         &nbs p;   TABLE ACCESS (CLUSTER) OF 'TS$'
16 15         &nbs p;     INDEX (UNIQUE SCAN) OF 'I_TS#' (NON- UNIQUE)
17  3         NEST ED LOOPS
18 17         &nbs p;   NESTED LOOPS
19 18         &nbs p;     TABLE ACCESS (FULL) OF 'TABPART$'
20 18         &nbs p;     TABLE ACCESS (BY INDEX ROWID) OF 'OB J$'
21 20         &nbs p;       INDEX (UNIQUE SCAN) OF 'I_OBJ1' (UNIQUE)
22 17         &nbs p;   TABLE ACCESS (CLUSTER) OF 'USER$'
23 22         &nbs p;     INDEX (UNIQUE SCAN) OF 'I_USER#' (NO N-UNIQUE)
24  3         NEST ED LOOPS
25 24         &nbs p;   NESTED LOOPS
26 25         &nbs p;     NESTED LOOPS
27 26         &nbs p;       TABLE ACCESS (FULL) OF 'USER$'
28 26         &nbs p;       TABLE ACCESS (BY INDEX ROWID) OF 'OBJ$'
29 28         &nbs p;         INDEX (R ANGE SCAN) OF 'I_OBJ2' (UNIQUE)
30 25         &nbs p;     TABLE ACCESS (BY INDEX ROWID) OF 'TA BCOMPART$'
31 30         &nbs p;       INDEX (UNIQUE SCAN) OF 'I_TABCOMPART$' (UNIQUE)
32 24         &nbs p;   TABLE ACCESS (CLUSTER) OF 'TS$'
33 32         &nbs p;     INDEX (UNIQUE SCAN) OF 'I_TS#' (NON- UNIQUE)
```

Statistics

```
----------------------------------------------------------
7          recursive calls
0          db block gets
20234      consistent gets
0          physical reads
0          redo size
724        bytes sent via SQL*Net to client
499        bytes received via SQL*Net from client
2          SQL*Net roundtrips to/from client
1          sorts (memory)
0          sorts (disk)
```

9 rows processed

Busquedas para demostrar que las tablas y particiones si tienen valores:

*select * from TAB_PAR_4 partition (SYS_P13);*

Pantalla de Trabajo

Archivo o URL: Examinar... No se ha seleccionado ningún archivo. Cargar Archivo de Comandos

Introducir Sentencias:

```
select * from TAB_PAR_4 partition (SYS_P13);
```

Ejecutar | Guardar Archivo de Comandos | Limpiar Pantalla | Cancelar

ID	INFO
6	AAAA6
11	AAAA11
13	AAAA13
27	AAAA27
30	AAAA30
34	AAAA34
40	AAAA40
46	AAAA46
50	AAAA50
53	AAAA53
54	AAAA54
55	AAAA55
61	AAAA61
74	AAAA74
ID	INFO
76	AAAA76
79	AAAA79
80	AAAA80
81	AAAA81
88	AAAA88
97	AAAA97
108	AAAA108
111	AAAA111
129	AAAA129

Vemos que si tiene valores dentro, pero por alguna razón no los muestra en el num_rows.

*select * from tab_par_5;*

Pantalla de Trabajo

Archivo o URL: Examinar... No se ha seleccionado ningún archivo. Cargar Archivo de Comandos

Introducir Sentencias:

```
select * from tab_par_5;
```

Ejecutar | Guardar Archivo de Comandos | Limpiar Pantalla | Cancelar

ID	INFO
6	AAAA6
11	AAAA11
30	AAAA30
34	AAAA34
46	AAAA46
54	AAAA54
55	AAAA55
74	AAAA74
81	AAAA81
88	AAAA88
97	AAAA97
108	AAAA108
111	AAAA111
133	AAAA133
ID	INFO
146	AAAA146
174	AAAA174
179	AAAA179
184	AAAA184

Igual que en lo anterior, vemos que si tiene valores dentro.

Los contamos para demostrar cuantos hay dentro de cada particion (en el caso de las particiones de la tabla de 4)

select count(*) from TAB_PART_4 partition (SYS_P13);

...

E igual en la tabla 5

select count(*) from TAB_PART_5 partition (SYS_P21);