INFORME DE TRABAJO 4 BASES DE DATOS II

INTEGRANTES:
Carlos Mario Chávez Aguilera
Eduardo Santos Ruiz

DOCENTE:
Francisco Javier Moreno

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MEDELLIN
ABRIL DE 2020

PUNTO 1: En este caso se usa la tabla venta sin indexado más allá del genérico de SQL.

Acerca de las consultas, estas trabajan en un sistema de ventas que son guardadas en una tabla SQL con 3 datos de entrada obligatorios: **codventa**, **idvendedor** y **unidades**. La función principal de todas las consultas es la identificar en que rango está cada vendedor dependiendo del valor unidades, siendo estos los niveles 1,2,3 y 4. Las consultas funcionan de diferentes formas dependiendo de cómo se interprete la tabla venta, estas serán llenadas con el siguiente código PL/SQL.

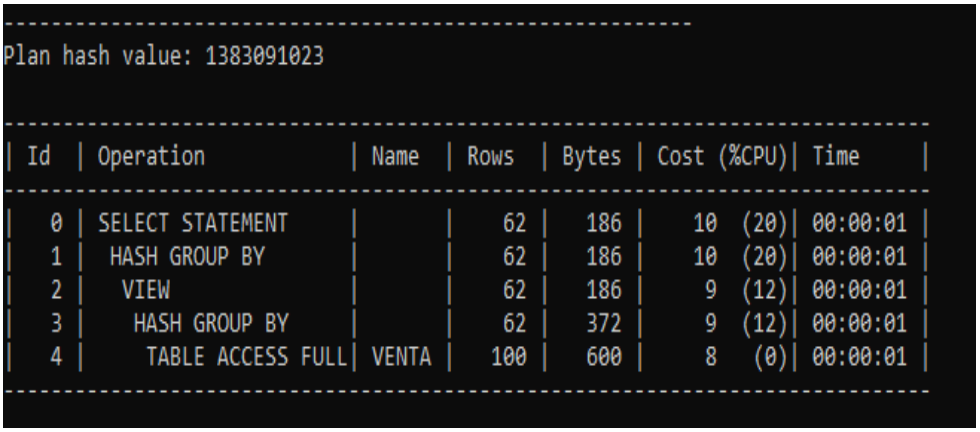
```
CREATE OR REPLACE PROCEDURE insercionDatos(x NUMBER) IS
    v venta%rowtype;
BEGIN
    FOR i IN 1..x LOOP
        v.codventa := i;
        v.idvendedor := dbms_random.value(1,x);
        v.unidades := dbms_random.value(1,x);
        INSERT INTO venta VALUES v;
    END LOOP;
    COMMIT;
END;
/
```

- La clave primaria será un valor consecutivo por lo que no será repetido.
- Si la cantidad de vendedores es baja con respecto a la muestra total (x) entonces todos tendrán muchas ventas y por lo tanto todos serán nivel 4. Si por el contrario son muy altas, las ventas estarán repartidas entre muchos vendedores distintos y difícilmente llegarían a nivel 4.
- De igual forma con las unidades, siendo estas las unidades vendidas en cada factura. Si el número de unidades vendidas es bajo, los vendedores tenderán a ser nivel 1. Por el contrario, si son mayores a 50 no habría ni un solo caso de nivel 1.
- Por lo anterior se decide qué número de vendedores y máximo de unidades vendidas sean iguales al tamaño de muestra. De esta forma se mantiene un equilibrio en el que se pueden observar muestras para todos los niveles en cada consulta.
- A mayor tamaño de muestra, mayor cantidad de vendedores de nivel 4.

Análisis de consultas: Para efectos de este trabajo, se considerarán todas las consultas equivalentes entre sí porque no se encontraron diferencias en los resultados para cada uno de las configuraciones o tamaños de muestra utilizados.

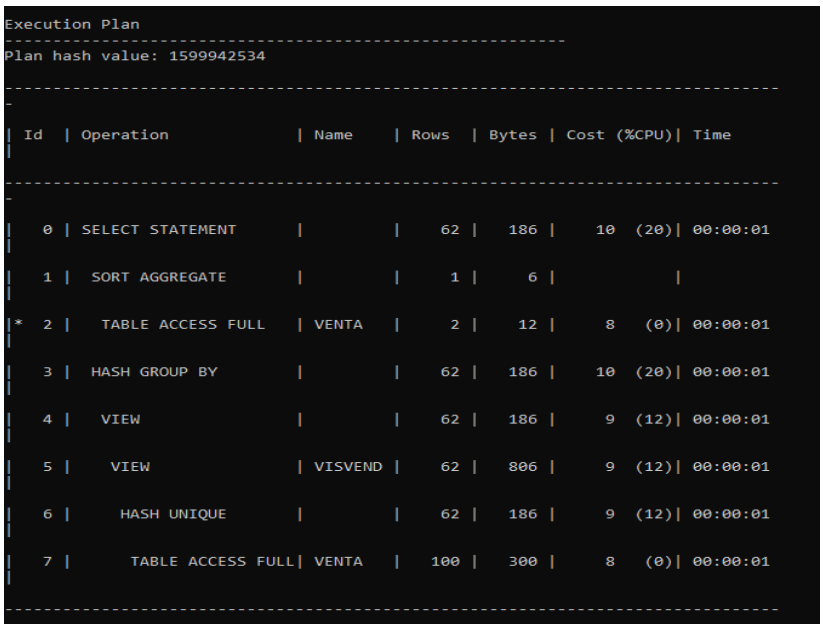
Todas las consultan se encargan de clasificar los vendedores a partir de cuantas unidades han vendido total. Esto valiéndose del hecho de que cada **idvendedor** identifica a un vendedor y unidades sería el número de ventas por cada factura para ese mismo vendedor. Al final todas las consultas imprimen cuantos vendedores hay en cada una de las 4 categorías. A continuación, el Explain Plan de cada consulta y una breve explicación del mismo:

EXPLAIN PLAN PARA CONSULTAS PUNTO 1:



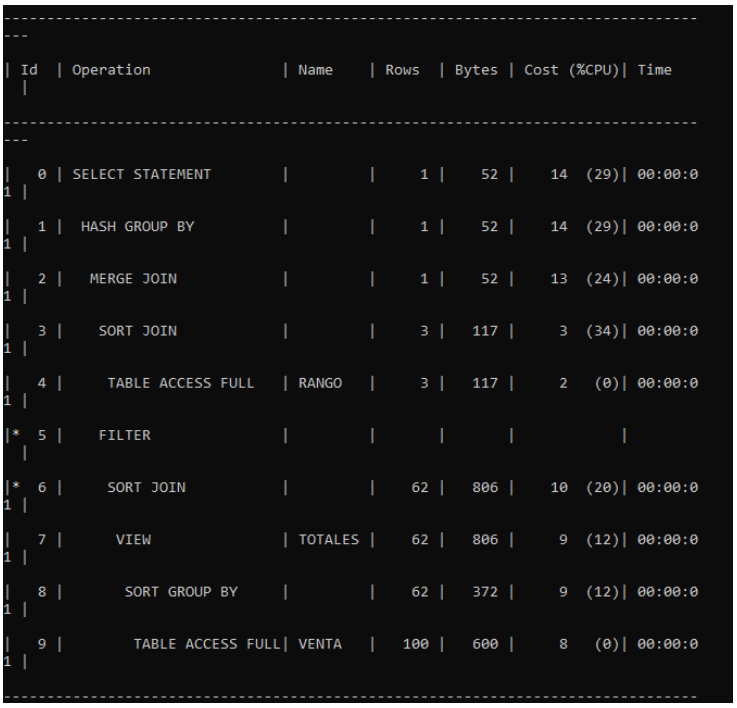
```
Plan hash value: 1383091023

| Id  | Operation           | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
|   0 | SELECT STATEMENT    |       |    62 |   186 |    10  (20)| 00:00:01 |
|   1 |  HASH GROUP BY      |       |    62 |   186 |    10  (20)| 00:00:01 |
|   2 |   VIEW              |       |    62 |   186 |     9  (12)| 00:00:01 |
|   3 |    HASH GROUP BY    |       |    62 |   372 |     9  (12)| 00:00:01 |
|   4 |     TABLE ACCESS FULL| VENTA |   100 |   600 |     8   (0)| 00:00:01 |
```

Consulta 1: Esta consulta agrupa los vendedores por **idvendedor.** Se clasifica cada vendedor a partir de la suma de sus unidades para todas sus facturas. Se categorizan los vendedores a partir de una clausula CASE. El Explain Plan indica que el recorrido de la tabla busca coincidencias TABLE ACCESS FULL.



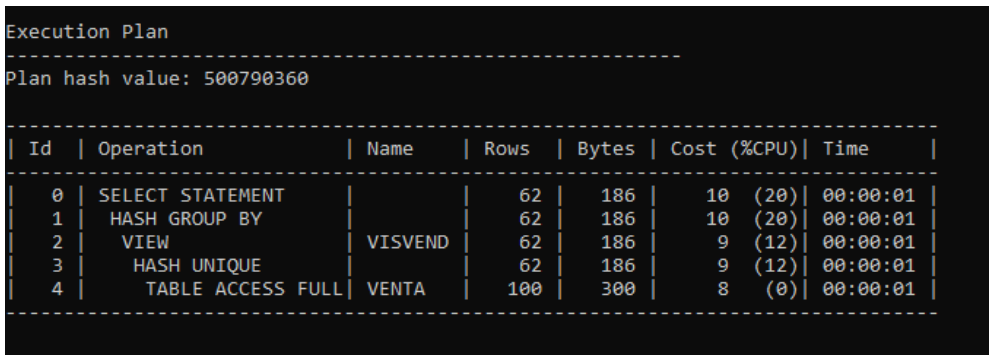
```
Execution Plan
Plan hash value: 1599942534

| Id  | Operation            | Name    | Rows  | Bytes | Cost (%CPU)| Time
|   0 | SELECT STATEMENT     |         |    62 |   186 |    10  (20)| 00:00:01
|   1 |  SORT AGGREGATE      |         |     1 |     6 |            |
|*  2 |   TABLE ACCESS FULL  | VENTA   |     2 |    12 |     8   (0)| 00:00:01
|   3 |  HASH GROUP BY       |         |    62 |   186 |    10  (20)| 00:00:01
|   4 |   VIEW               |         |    62 |   186 |     9  (12)| 00:00:01
|   5 |    VIEW              | VISVEND |    62 |   806 |     9  (12)| 00:00:01
|   6 |     HASH UNIQUE      |         |    62 |   186 |     9  (12)| 00:00:01
|   7 |      TABLE ACCESS FULL| VENTA   |   100 |   300 |     8   (0)| 00:00:01
```

Consulta 2: Primeramente, es necesario la creación de una vista llamada **visvend** en la que se definen los distintos **idvendedor** que hay a partir de una clausula **DISTINCT(HASH UNIQUE).** Como en la consulta 1 se usa la cláusula CASE para las clasificaciones, pero esta vez se usa una variable **sumin** que recorre la vista **visvend** creada inicialmente. Es buena idea porque el recorrido de la vista puede ser más eficiente que el recorrido de la tabla.



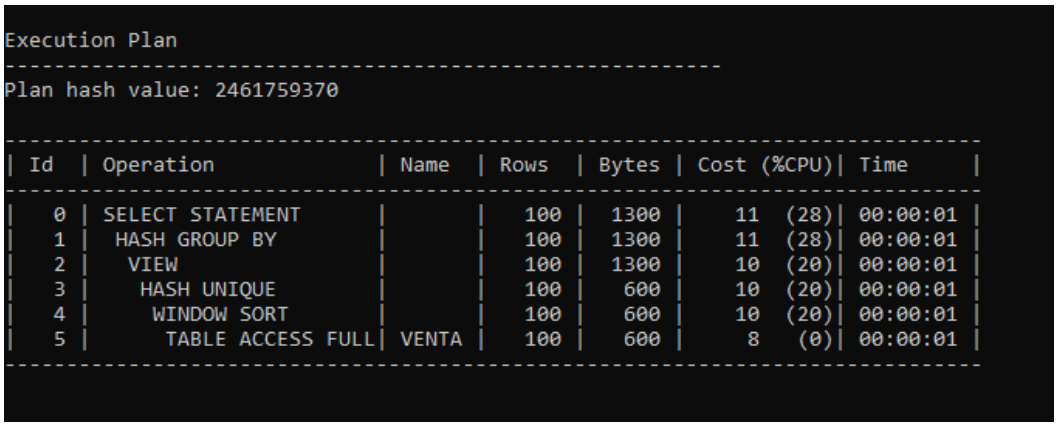
```
-------------------------------------------------------------------------------
---
| Id  | Operation              | Name    | Rows  | Bytes | Cost (%CPU)| Time
  |
-------------------------------------------------------------------------------
---
|   0 | SELECT STATEMENT       |         |     1 |    52 |    14  (29)| 00:00:0
1 |
|   1 |  HASH GROUP BY         |         |     1 |    52 |    14  (29)| 00:00:0
1 |
|   2 |   MERGE JOIN           |         |     1 |    52 |    13  (24)| 00:00:0
1 |
|   3 |    SORT JOIN           |         |     3 |   117 |     3  (34)| 00:00:0
1 |
|   4 |     TABLE ACCESS FULL  | RANGO   |     3 |   117 |     2   (0)| 00:00:0
1 |
|*  5 |    FILTER              |         |       |       |            |
  |
|*  6 |     SORT JOIN          |         |    62 |   806 |    10  (20)| 00:00:0
1 |
|   7 |      VIEW              | TOTALES |    62 |   806 |     9  (12)| 00:00:0
1 |
|   8 |       SORT GROUP BY    |         |    62 |   372 |     9  (12)| 00:00:0
1 |
|   9 |        TABLE ACCESS FULL| VENTA   |   100 |   600 |     8   (0)| 00:00:0
1 |
-------------------------------------------------------------------------------
```

Consulta 3: Esta consulta se encarga de crear una tabla llamada **rango** donde se insertan algunas filas que contienen información para definir los tipos de vendedores. Esta consulta propone que se busque en tabla **rango** cada vez que se vaya a clasificar un vendedor. Crea dos vistas adicionales, una llamada **totales** para obtener el identificador de cada vendedor con su respectiva suma. La segunda vista agrupa la tabla de rango y la vista totales llamada **clases**. La consulta final llama a la vista **clases** antes que a la tabla rango o venta.



```
Execution Plan
----------------------------------------------------------
Plan hash value: 500790360

-------------------------------------------------------------------------------
| Id  | Operation            | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |         |    62 |   186 |    10  (20)| 00:00:01 |
|   1 |  HASH GROUP BY       |         |    62 |   186 |    10  (20)| 00:00:01 |
|   2 |   VIEW               | VISVEND |    62 |   186 |     9  (12)| 00:00:01 |
|   3 |    HASH UNIQUE       |         |    62 |   186 |     9  (12)| 00:00:01 |
|   4 |     TABLE ACCESS FULL| VENTA   |   100 |   300 |     8   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Consulta 4: Usa al igual que la consulta 2, la VIEW **visvend** es usada para distinguir todos los vendedores que existen en la tabla venta. En esta consulta se crea la función **clase** con parámetro de entrada **idvendedor** encargado del recorrido de tabla. En Explain se evidencia que antes entra a la tabla Venta que a visvend.



```
Execution Plan
----------------------------------------------------------
Plan hash value: 2461759370

-------------------------------------------------------------------------------
| Id  | Operation            | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |       |   100 |  1300 |    11  (28)| 00:00:01 |
|   1 |  HASH GROUP BY       |       |   100 |  1300 |    11  (28)| 00:00:01 |
|   2 |   VIEW               |       |   100 |  1300 |    10  (20)| 00:00:01 |
|   3 |    HASH UNIQUE       |       |   100 |   600 |    10  (20)| 00:00:01 |
|   4 |     WINDOW SORT      |       |   100 |   600 |    10  (20)| 00:00:01 |
|   5 |      TABLE ACCESS FULL| VENTA |   100 |   600 |     8   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Consulta 5: Difiere de la anterior porque no se usa VIEW en ninguna parte a pesar de usar igualmente una función llamada clasin. La diferencia principal está en que el operador Distinct se ejecuta en la consulta final y no una view aparte. Está agrupada nuevamente por sv.

Todas las consultas anteriores son diferentes formas de llegar al mismo resultado. Por lo tanto, son equivalentes.

TKPROF PARA CONSULTAS PUNTO 1:

A continuación, se analizarán las consultas a partir de cuantos datos fueron insertados. Algunas consideraciones:

- Tiempo: Tiempo que tardó la consulta en segundos.
- CPU: Tiempo de procesador usado en segundos.
- Tkprof1: LIOs (f+g) sobre filas procesadas (h).
- TKprof2: Filas retornadas (i) sobre traídas (*fetches*) (j).
- Tkprof3: Lecturas de disco (k) sobre LIOs (f+g).
- El valor de TKProf2 siempre será 2 porque el formato de impresión de las consultas es igual para todos los casos (4 filas y 2 columnas).
- Si algún dato en la tabla es 0, es porque su magnitud no es considerable.
- En los anexos de este trabajo se pueden encontrar las tablas con datos completos, los resultados de TKProf y Explain para cada consulta y tamaño de muestra.
- No sé probó con un número de datos insertados superior a 1.000.000 en los dos puntos debido a que las condiciones técnicas no lo permitieron.

EXECUTE insercionDatos(100)

		Consulta1	Consulta2	Consulta3	Consulta4	Consulta5
100	Tiempo	0	0,23	0	0,1	0,01
	CPU	0	0	0	0	0
	TkProf1	689,25	40665,75	690	689,25	689,25
	TkProf2	2	2	2	2	2
	TkProf3	0%	0%	0%	0%	0%

Se comienza con 100 datos insertados y ya en esta consulta se dan pistas acerca de su funcionamiento. Ninguna cumple con lo óptimo para el parámetro TKProf1 (<20).

EXECUTE insercionDatos(100000)

		Consulta1	Consulta2	Consulta3	Consulta4	Consulta5
100.000	Tiempo	0,07	441,89	0,12	6,42	0,23
	CPU	0,07	437,57	0,12	3,76	0,23
	TKProf1	689,25	43477200,8	690	689,25	689,25
	TKProf2	2	2	2	2	2
	TKProf3	2%	0%	0%	0%	0%

Con esta cantidad de datos es posible observar diferencias considerables entre las consultas, por una parte, las consultas 1, 3 y 5 son casi instantáneas, por el contrario, las consultas 2 y 4 requieren de segundos o minutos para ser llevadas a cabo. En el caso de la Consulta 2 puede deberse al alto número de queries que se hacen al sistema para resolver la consulta, incluso más que la cantidad de datos de la muestra.

Según los intervalos de buen rendimiento para el parámetro TKProf1, este debería estar por debajo de 20 y no es el caso en ninguna de las consultas, pero es preocupante el exorbitante dato en la consulta 2. Además, se puede observar como la mayoría de tiempo que tardó la Consulta 2, esta fue hecha por la CPU. Debido a que el tiempo de CPU es muy cercano al ELAPSED, quiere decir que el sistema no estuvo mucho tiempo en espera innecesaria. Aun así, en comparación con las demás consultas, el tiempo de ejecución es demasiado alto. Es necesario mucho tiempo de procesador para una simple consulta, indica un alto consumo de recursos en el sistema, un proceso poco eficiente, es claramente una consulta que requeriría de urgente afinamiento en caso de estar en uso.

El TKprof3 no es preocupante para ninguna de las consultas con este número de muestra, pero es la primera vez que un dato es considerable, tal es el caso de la Consulta 1 que tuvo que llamar 45 veces a disco (2%). A pesar de estar por debajo del 10% recomendable, esta consulta empieza a llamar información del disco con tan solo 100.000 datos. Esto podría ser indicio de mala escalabilidad para la Consulta1.

EXECUTE insercionDatos(1000000)

		Consulta1	Consulta2	Consulta3	Consulta4	Consulta5
1.000.000	Tiempo	0,9	18759,89	2,06	108,43	2,43
	CPU	0,67	18755,82	2,01	80,46	2,43
	TkProf1	1034	434585520	689,25	688,75	688,25
	TkProf2	2	2	2	2	2
	TkProf3	12%	0%	0%	0%	0%

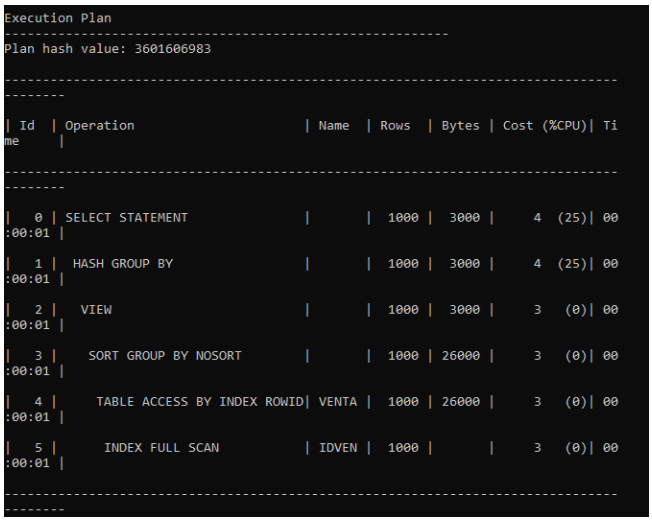
Esta es la máxima cantidad de datos que fueron probados. Nuevamente la consulta 2 presentó problemas de rendimiento como era de esperarse, demorando más de 5 horas en realizar su proceso, La consulta 4 también tardó más de 2 horas en presentar resultados, el dato de 108 segundos entregado por TKProf no corresponde con la realidad, se desconoce el porqué de dicho resultado. Siendo que los datos no son fidedignos para esta consulta, no se pueden hacer conclusiones realistas, pero es probable que el tiempo de CPU sea mucho menor al ELAPSED, esto indicaría tiempos muertos dentro de la consulta en espera de otros procesos para continuar, ahí podría radicar su ineficiencia. Como se dice, no se puede afirmar nada, pero es una posibilidad considerando además que era una tendencia con menor muestra de datos.

Por otra parte, el problema de lectura de disco para la Consulta 1 ahora es mayor a 10% y está por fuera de los parámetros recomendados, a pesar de que su tiempo de respuesta es el mejor para todos los casos. Esto podría indicar que es una consulta rápida con altos requerimientos de sistema pero que suple la necesidad siempre y cuando se tengan los recursos.

Para ser más concluyentes con las consultas 3 y 5, sería necesaria una mayor cantidad de datos en tabla. Están correctas dentro de los criterios adecuados para TKProf 2 y TKProf 3. Se mantuvieron estables en la medida de TKProf1 a pesar de ser elevada desde un principio y los tiempos en arrojar resultados fueron muy buenos, cercanos ambas a los 2 segundos. Tienen buena escalabilidad, por lo menos hasta este punto, así como casi nulo tiempo de espera del CPU.

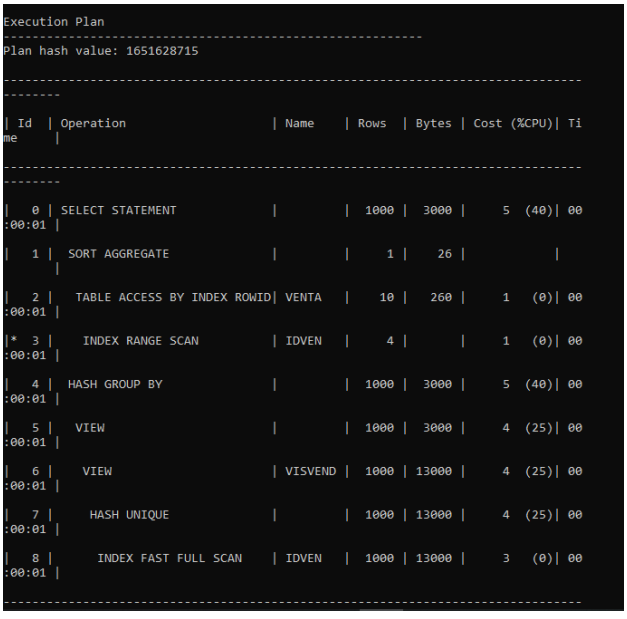
PUNTO 2: Usando el indexado idven para la tabla venta.

EXPLAIN PLAN PARA CONSULTAS PUNTO 2:



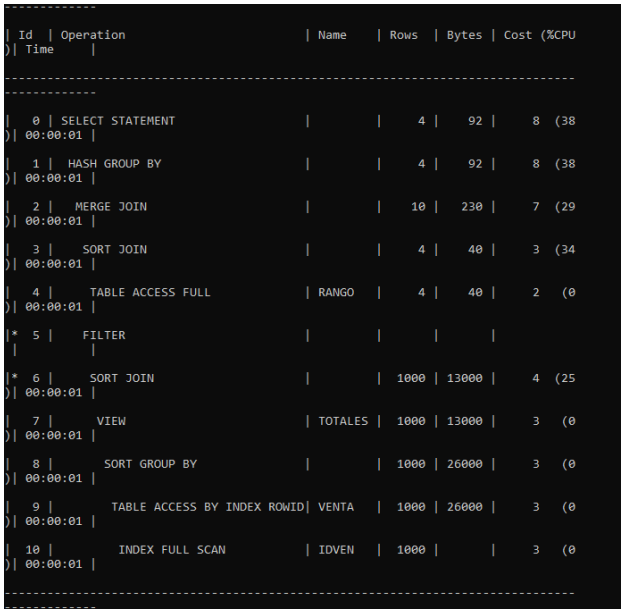
```
Execution Plan
----------------------------------------------------------
Plan hash value: 3601606983

-------------------------------------------------------------------------------
--------

| Id  | Operation                     | Name  | Rows  | Bytes | Cost (%CPU)| Ti
me     |

-------------------------------------------------------------------------------
--------

|   0 | SELECT STATEMENT              |       |  1000 |  3000 |     4  (25)| 00
:00:01 |

|   1 |  HASH GROUP BY                |       |  1000 |  3000 |     4  (25)| 00
:00:01 |

|   2 |   VIEW                        |       |  1000 |  3000 |     3   (0)| 00
:00:01 |

|   3 |    SORT GROUP BY NOSORT       |       |  1000 | 26000 |     3   (0)| 00
:00:01 |

|   4 |     TABLE ACCESS BY INDEX ROWID| VENTA |  1000 | 26000 |     3   (0)| 00
:00:01 |

|   5 |      INDEX FULL SCAN          | IDVEN |  1000 |       |     3   (0)| 00
:00:01 |

-------------------------------------------------------------------------------
--------
```

Consulta 1: Al ser indexada la tabla venta, el proceso para hallar una solución cambia. Antes la consulta era resuelta con HASH GROUB BY, mientras que ahora lo hace por SORT GROUP BY, esta última usa el indexado idven. Hay una diferencia fundamental y es la forma en que son accedidos los datos, para esta consulta serán tomados a partir del proceso INDEX FULL SCAN con un costo de 3, mejorando el TABLE ACCESS FULL con 8. Esto implica un costo muchísimo menor para la consulta en el global. Más adelante en el análisis con TKProf será todavía más evidenciado. En general con mejor rendimiento que la Consulta 1 original.



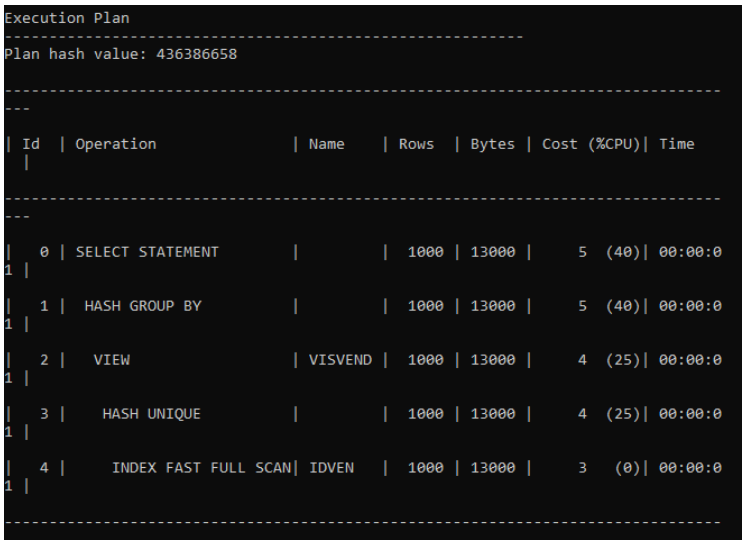
```
Execution Plan
----------------------------------------------------------
Plan hash value: 1651628715

-------------------------------------------------------------------------------
--------

| Id  | Operation                     | Name    | Rows  | Bytes | Cost (%CPU)| Ti
me     |

-------------------------------------------------------------------------------
--------

|   0 | SELECT STATEMENT              |         |  1000 |  3000 |     5  (40)| 00
:00:01 |

|   1 |  SORT AGGREGATE               |         |     1 |    26 |            |
       |

|   2 |   TABLE ACCESS BY INDEX ROWID| VENTA    |    10 |   260 |     1   (0)| 00
:00:01 |

|*  3 |    INDEX RANGE SCAN           | IDVEN   |     4 |       |     1   (0)| 00
:00:01 |

|   4 |  HASH GROUP BY                |         |  1000 |  3000 |     5  (40)| 00
:00:01 |

|   5 |   VIEW                        |         |  1000 |  3000 |     4  (25)| 00
:00:01 |

|   6 |    VIEW                       | VISVEND |  1000 | 13000 |     4  (25)| 00
:00:01 |

|   7 |     HASH UNIQUE               |         |  1000 | 13000 |     4  (25)| 00
:00:01 |

|   8 |      INDEX FAST FULL SCAN     | IDVEN   |  1000 | 13000 |     3   (0)| 00
:00:01 |

-------------------------------------------------------------------------------
```

Consulta 2: Esta fue la consulta que más se benefició por el indexado, esto debido a que accedía dos veces a la tabla venta con TABLE ACCESS FULL(8) trayendo muchísimo costo a la consulta. Son reemplazados por INDEX FAST SCAN (3) y INDEX RANGE SCAN(1). Se reduce a la mitad el costo de la operación, sin dejar de ser una consulta muy mejorable.



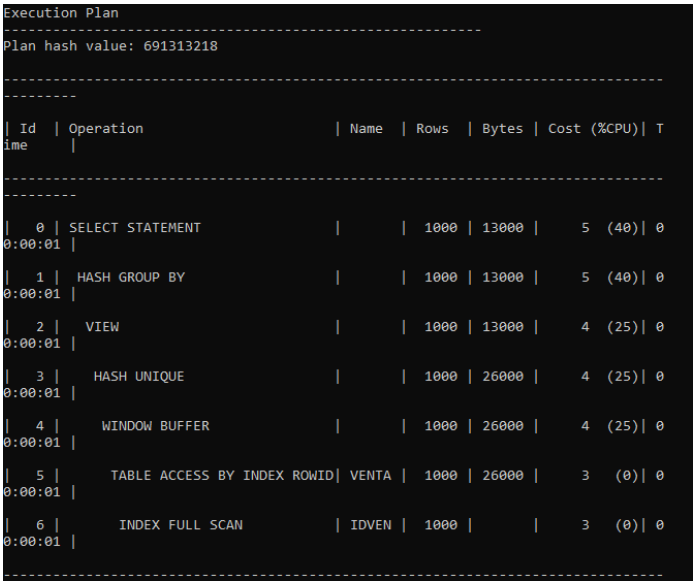
```
-------------
| Id  | Operation                      | Name    | Rows  | Bytes | Cost (%CPU
)| Time     |
-------------------------------------------------------------------------------
-------------
|   0 | SELECT STATEMENT               |         |     4 |    92 |     8  (38
)| 00:00:01 |
|   1 |  HASH GROUP BY                 |         |     4 |    92 |     8  (38
)| 00:00:01 |
|   2 |   MERGE JOIN                   |         |    10 |   230 |     7  (29
)| 00:00:01 |
|   3 |    SORT JOIN                   |         |     4 |    40 |     3  (34
)| 00:00:01 |
|   4 |     TABLE ACCESS FULL          | RANGO   |     4 |    40 |     2   (0
)| 00:00:01 |
|*  5 |    FILTER                      |         |       |       |
 |          |
|*  6 |     SORT JOIN                  |         |  1000 | 13000 |     4  (25
)| 00:00:01 |
|   7 |      VIEW                      | TOTALES |  1000 | 13000 |     3   (0
)| 00:00:01 |
|   8 |       SORT GROUP BY            |         |  1000 | 26000 |     3   (0
)| 00:00:01 |
|   9 |        TABLE ACCESS BY INDEX ROWID| VENTA   |  1000 | 26000 |     3   (0
)| 00:00:01 |
|  10 |         INDEX FULL SCAN        | IDVEN   |  1000 |       |     3   (0
)| 00:00:01 |
-------------------------------------------------------------------------------
-------------
```

Consulta 3: La estructura de la consulta es diferente a las otras porque accede a dos tablas: Rango y Venta. Como en los otros casos cambia el procedimiento para acceder a tabla Venta debido al indexado, pero no es el caso con Rango. Esta última no tiene un indexado por lo que se seguirá accediendo con el método TABLE FULL ACCES. Esto realmente no es problema para este caso porque la tabla Rango no tiene una gran cantidad de datos.



```
Execution Plan
----------------------------------------------------------
Plan hash value: 436386658

-------------------------------------------------------------------------------
---
| Id  | Operation              | Name    | Rows  | Bytes | Cost (%CPU)| Time
  |
-------------------------------------------------------------------------------
---
|   0 | SELECT STATEMENT       |         |  1000 | 13000 |     5  (40)| 00:00:0
1 |
|   1 |  HASH GROUP BY         |         |  1000 | 13000 |     5  (40)| 00:00:0
1 |
|   2 |   VIEW                 | VISVEND |  1000 | 13000 |     4  (25)| 00:00:0
1 |
|   3 |    HASH UNIQUE         |         |  1000 | 13000 |     4  (25)| 00:00:0
1 |
|   4 |     INDEX FAST FULL SCAN| IDVEN   |  1000 | 13000 |     3   (0)| 00:00:0
1 |
-------------------------------------------------------------------------------
```

Consulta 4: Para las consultas 2 y 4 que usaron una tabla externa VIEW **visvend** para acceder a la tabla venta, igualmente se ven beneficiados porque la VIEW también usa INDEX FAST FULL SCAN.



```
Execution Plan
----------------------------------------------------------
Plan hash value: 691313218

-------------------------------------------------------------------------------
---------

| Id  | Operation                     | Name  | Rows  | Bytes | Cost (%CPU)| T
ime     |

-------------------------------------------------------------------------------
---------

|   0 | SELECT STATEMENT              |       |  1000 | 13000 |     5  (40)| 0
0:00:01 |

|   1 |  HASH GROUP BY                |       |  1000 | 13000 |     5  (40)| 0
0:00:01 |

|   2 |   VIEW                        |       |  1000 | 13000 |     4  (25)| 0
0:00:01 |

|   3 |    HASH UNIQUE                |       |  1000 | 26000 |     4  (25)| 0
0:00:01 |

|   4 |     WINDOW BUFFER             |       |  1000 | 26000 |     4  (25)| 0
0:00:01 |

|   5 |      TABLE ACCESS BY INDEX ROWID| VENTA |  1000 | 26000 |     3   (0)| 0
0:00:01 |

|   6 |       INDEX FULL SCAN         | IDVEN |  1000 |       |     3   (0)| 0
0:00:01 |

-------------------------------------------------------------------------------
```

TKPROF PARA CONSULTAS PUNTO 2:

EXECUTE insercionDatos(100)

	Consulta1	Consulta2	Consulta3	Consulta4	Consulta5
Tiempo	0	0	0	0	0
CPU	0	0	0	0	0
TkProf1	1,25	2,75	2,5	1	1,25
TkProf2	2	2	2	2	2
TkProf3	0%	0%	0%	0%	0%

Para sorpresa propia, los parámetros de TKProf1 fueron solucionados para todas las consultas una vez la tabla Venta fue indexada., todos los resultados están por debajo de 20, el valor recomendado. Esto puede deberse a que las consultas recorren la tabla ventas por medio del indexado y no por el recorrido natural de la misma. Esto reduce drásticamente la cantidad de queries que se hacen a la base de datos.

EXECUTE insercionDatos(10000)

	Consulta1	Consulta2	Consulta3	Consulta4	Consulta5
Tiempo	0,01	0,02	0,01	0,24	0,03
CPU	0,01	0,01	0,01	0,23	0,03
TkProf1	2406	2784,75	2399,25	8,25	2398,75
TkProf2	2	2	2	2	2
TkProf3	0%	0%	0%	0%	0%

En este punto datos de TKProf1 saltan hasta cerca de los 2500 para todos los casos excepto la Consulta 4 que crece más lento. Ya se evidencian una clara mejora en los tiempos para las Consultas 2 y Consulta 4. Esto puede deberse a que están recorriendo la tabla venta por medio de un índice y no genéricamente. Las diferencias en valores de costo evidenciado en el Explain Plan ahora es la explicación de la mejora radical de tiempo y menor cantidad de queries que se hacen por consulta.

EXECUTE insercionDatos(1000000)

	Consulta1	Consulta2	Consulta3	Consulta4	Consulta5
Tiempo	2,91	3,75	1,78	24,32	4,73
CPU	1,81	2,96	1,78	24,59	3,31
TkProf1	251988,25	323491,25	250719,75	818,75	250719,25
TkProf2	2	2	2	2	2
TkProf3	0,1%	0,1%	0,0%	0,0%	0,2%

Se puede distinguir un patrón en los resultados del parámetro de TKProf 1. Son de crecimiento lineal y proporcional al número de datos que está evaluando, para entender esto último se recomienda analizar el crecimiento en la tabla anexa de datos en EXCEL. Aun así, son valores pocos aceptables.

La consulta 1 soluciona su problema de lectura de disco hasta un porcentaje poco importante. Esto debido a dos cosas, son menos las veces que el computador tuvo que buscar en disco (la mitad) y que la cantidad de queries fue muchísimo mayor.

La consulta 2 reduce su tiempo de forma considerable, siendo esta la mayor evidencia del poder que tienen los índices para mejorar las búsquedas. Se redujo un tiempo de más de 5 horas a tan solo 4 segundos, ¡impresionante!

CONCLUSIONES:

- Es evidente como hubo una clara mejora de los tiempos en las consultas con la tabla Venta indexada. Evita que las consultas realicen procesos iterativos innecesarios. Siempre será más fácil buscar un índice que una coincidencia de valor.
- Es difícil elegir cuál de todas las consultas es la más eficiente u óptima, pero diría que sin indexar las mejores elecciones serían la Consulta 3 o 5 ya que tienen resultados muy parecidos y fueron los mejores. Personalmente iría por la 5
- Para las consultas con tabla indexada, la mejor opción sería la Consulta 3 ya que tiene el mejor tiempo y no tiene peticiones hechas al disco duro.
- Acerca de la decisión de si indexar o no la tabla, se recomienda hacerlo debido a que obtiene mejores resultados en los tiempos. Pero tendríamos que tener presente que la cantidad de peticiones para muestras de datos descomunalmente grandes podría ser perjudicial para el rendimiento. Todo dependerá entonces del tipo de máquina en la que opere la base de datos.