

Iteración 4

Carlos González. David Patiño. C11

Escogencia de índices:

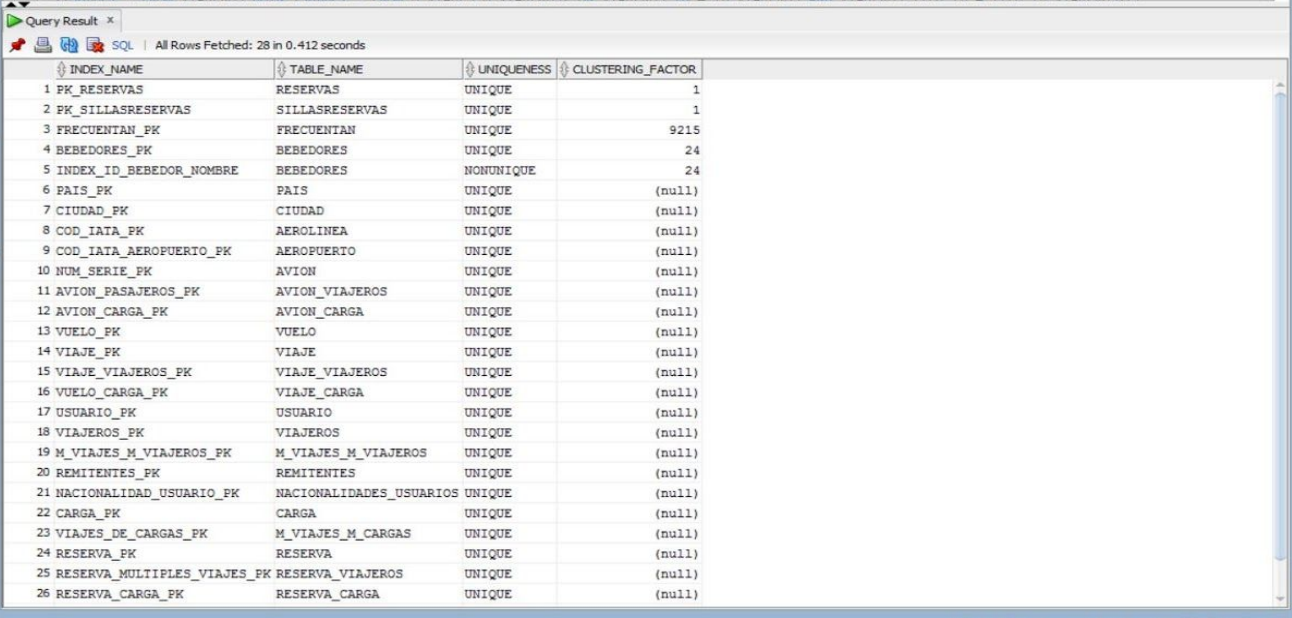
RFC7 Y RFC8:

- Vuelos.tipo: para esta parte del requerimiento será utilizado un BITMAP en la columna TIPO de la tabla de VUELO, ya que al haber solo dos opciones en esta va a permitir un filtrado rápido de la información. Adicionalmente, los costos de actualizar el índice no se verán afectados debido a que sus valores no son dinámicos, otro factor importante es que el espacio que ocupa este índice es menor debido a la baja cardinalidad de los datos.
- Vuelos.fecha: se considera que para este requerimiento se utilizará un B+ sobre la columna de FECHA en VIAJES, para poder facilitar el filtrado de los datos en un rango seleccionado, los datos al tener un diseño específico tienden a distribuirse proporcionalmente. El único problema en la implementación de B+ es que en el RFC8 al la consulta ser excluyente puede llegar a ser más extensa.
- Vuelos.Aerolinea: Un índice B+ sobre la columna AEROLINEA en la tabla de VUELO sería el índice más óptimo, ya que se necesita un filtrado rápido de la información y este índice es el que más reduce el costo de esta operación, además permite que no sea necesario ingresar a toda la tabla.

RFC9: para este requerimiento se pensó en que la utilización de índice B+ en la lista RESERVA_VIAJEROS en la columna de ID_VUELO sería lo más óptimo, ya que en la transacción al ser utilizada esta columna en numerosos joins el costo del mismo sería razonable.

RFC10: para facilitar la unión de tablas se pensó que utilizar un árbol B+ en la tabla VIAJE en la columna AVION es lo más eficaz, ya que la tabla es dinámica y cualquier otro índice resulta más costoso la inserción de tuplas.

Índices generados por Oracle:



Query Result: All Rows Fetched: 28 in 0.412 seconds

| INDEX_NAME | TABLE_NAME | UNIQUENESS | CLUSTERING_FACTOR |
|--------------------------------|-------------------------|------------|-------------------|
| 1 FK_RESERVAS | RESERVAS | UNIQUE | 1 |
| 2 FK_SILLASRESERVAS | SILLASRESERVAS | UNIQUE | 1 |
| 3 FRECUENTIAN_PK | FRECUENTIAN | UNIQUE | 9215 |
| 4 BEBEDORES_PK | BEBEDORES | UNIQUE | 24 |
| 5 INDEX_ID_BEBEDOR_NOMBRE | BEBEDORES | NONUNIQUE | 24 |
| 6 PAIS_PK | PAIS | UNIQUE | (null) |
| 7 CIUDAD_PK | CIUDAD | UNIQUE | (null) |
| 8 COD_IATA_PK | AEROLINEA | UNIQUE | (null) |
| 9 COD_IATA_AEROPUERTO_PK | AEROPUERTO | UNIQUE | (null) |
| 10 NUM_SERIE_PK | AVION | UNIQUE | (null) |
| 11 AVION_PASAJEROS_PK | AVION_VIAJEROS | UNIQUE | (null) |
| 12 AVION_CARGA_PK | AVION_CARGA | UNIQUE | (null) |
| 13 VUELO_PK | VUELO | UNIQUE | (null) |
| 14 VIAJE_PK | VIAJE | UNIQUE | (null) |
| 15 VIAJE_VIAJEROS_PK | VIAJE_VIAJEROS | UNIQUE | (null) |
| 16 VUELO_CARGA_PK | VIAJE_CARGA | UNIQUE | (null) |
| 17 USUARIO_PK | USUARIO | UNIQUE | (null) |
| 18 VIAJEROS_PK | VIAJEROS | UNIQUE | (null) |
| 19 M_VIAJES_M_VIAJEROS_PK | M_VIAJES_M_VIAJEROS | UNIQUE | (null) |
| 20 REMITENTES_PK | REMITENTES | UNIQUE | (null) |
| 21 NACIONALIDAD_USUARIO_PK | NACIONALIDADES_USUARIOS | UNIQUE | (null) |
| 22 CARGA_PK | CARGA | UNIQUE | (null) |
| 23 VIAJES_DE_CARGAS_PK | M_VIAJES_M_CARGAS | UNIQUE | (null) |
| 24 RESERVA_PK | RESERVA | UNIQUE | (null) |
| 25 RESERVA_MULTIPLES_VIAJES_PK | RESERVA_VIAJEROS | UNIQUE | (null) |
| 26 RESERVA_CARGA_PK | RESERVA_CARGA | UNIQUE | (null) |

Line 7 Column 1 | Insert | Modified | Windows: C

Como se ve en la captura de pantalla todos los índices que son generados por Oracle son asignados al agregar una llave primaria a una tabla. Las características principales de estos índices es que son únicos y primarios, por lo tanto son los únicos en tener esta característica. Los índices que más ayudan a los requerimientos funcionales son :

RFC7 y RFC8:

- VIAJE_PK: este índice permite una unión rápida con la tabla de VUELOS, lo cual es importante para el RFC7 y RFC8, para así saber la información de los dos.
- VUELO_PK: este índice permite una unión rápida con la tabla VIAJES, lo cual es importante para el RFC7 y RFC8, para así saber la información de los dos.

RFC9:

- RESERVA_PK: este índice es indispensable para las consultas en el RFC9, ya que permite calcular el número de sillitas de un avión, además de conectarse con avión y usuario.

RFC10:

- AVION_PK: este índice permite la unión entre viaje y reserva. Esto ayuda a las consultas del RFC10.

Análisis para cada requerimiento:

RFC7 :

Sentencia SQL:

```
SELECT * FROM ISIS2304A131620.VUELO JOIN ISIS2304A131620.VIAJE ON
ISIS2304A131620.VUELO.ID= ISIS2304A131620.VIAJE.ID_VUELO WHERE
((ISIS2304A131620.VUELO.AEROPUERTO_SALIDA = "" + aeropuerto+ "" OR
ISIS2304A131620.VUELO.AEROPUERTO_LLEGADA = ""+aeropuerto+"")AND
ISIS2304A131620.VIAJE.AVION = ""+aeronave+"") ORDER BY "+organizacion;
```

Distribución de los datos:

- Aeropuerto: Considerando que existen 5000 aeropuertos en la base de datos, y aunque el tráfico de viajes de un aeropuerto depende de su importancia (que no está parametrizada específicamente en la base de datos), al solo elegirse un aeropuerto para la consulta es sensato pensar en una distribución cercana a .2% de los datos, lo cual es 1/5000
- Hora llegada y salida: Considerando que el requerimiento especifica que estos parámetros deben ser comparados por igualdad en la consulta, la selectividad puede llegar a ser muy alta, pues existen 1000 combinaciones diferentes del tipo “hh:mm”, y por ello sería una distribución de 1/1000 = 0.01%
- Rango de fechas: La distribución de viajes por fechas, en principio, es equitativa a lo largo de las semanas, por la frecuencia que tiene cada vuelo en días a la semana; Sin embargo, al momento de hacer una consulta la selectividad se verá definida

Tipo aeronave: Considerando que las aeronaves pueden ser “comerciales” o “de carga”, realmente la distribución de datos dependerá más de cuántos vuelos son efectivamente de carga o comerciales. Las alternativas del parámetro son de carácter binario, se puede estimar una selectividad cercana a 50%.

- Aerolínea: En la base de datos existen 10 aerolíneas, por lo que no es un factor muy relevante para las consultas.

Plan de consulta:

| OPERATION | OBJECT_NAME | CARDINALITY | COST |
|---|-------------|-------------|------|
| SELECT STATEMENT | | | 2431 |
| SORT (ORDER BY) | | 45 | 2431 |
| HASH JOIN | | 45 | 2430 |
| Access Predicates VUELO.ID=VIAJE.ID_VUELO | | | |
| NESTED LOOPS | | 45 | 2430 |
| NESTED LOOPS | | | |
| STATISTICS COLLECTOR | | | |
| TABLE ACCESS (FULL) VIAJE | VIAJE | 1510 | 2158 |
| Filter Predicates VIAJE.AVION=Avion | | | |
| INDEX (UNIQUE SCAN) VUELO_PK | VUELO_PK | | |
| Access Predicates VUELO.ID=VIAJE.ID_VUELO | | | |
| TABLE ACCESS (BY INDEX ROWID) VUELO | VUELO | 1 | 273 |
| Filter Predicates VUELO.AEROPUERTO_SA VUELO.AEROPUERTO_LL | | | |
| TABLE ACCESS (FULL) VUELO | VUELO | 2921 | 273 |
| Filter Predicates VUELO.AEROPUERTO_SALID VUELO.AEROPUERTO_LLEGA | | | |
| Other XML | | | |
| (info) | | | |

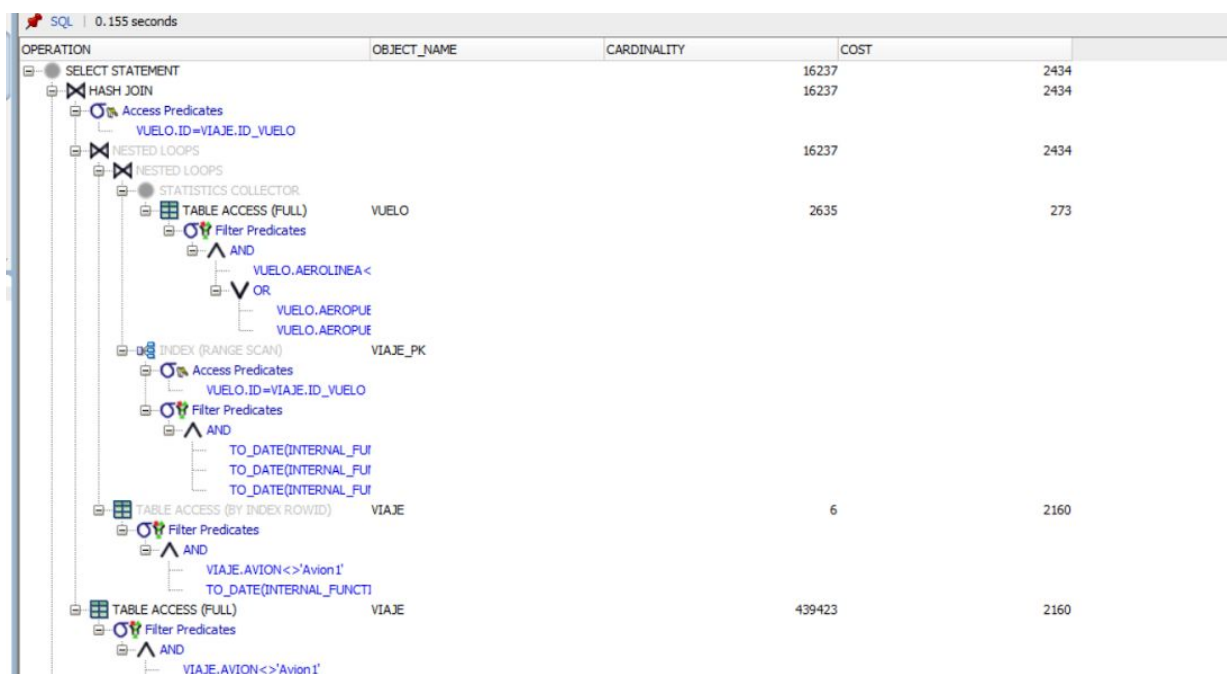
Oracle logra reducir los costos considerablemente con el uso de accesos a la tabla mediante index rowid, donde se aprovecha de los índices establecidos en los vuelos para no tener que recorrer toda la tabla.

```
SELECT * FROM VUELO JOIN ISIS2304A131620.VIAJE ON VUELO.ID= VIAJE.ID_VUELO
WHERE (((VUELO.AEROPUERTO_SALIDA = 'Aeropuerto1' OR
VUELO.AEROPUERTO_LLEGADA = 'Aeropuerto1') AND TO_DATE(HORA_SALIDA) >=
TO_DATE('11-11-2016','DD-MM-YYYY') AND TO_DATE(HORA_SALIDA) <=
TO_DATE('11-11-2017','DD-MM-YYYY')) AND VUELO.AEROLINEA!='AV' AND
VIAJE.AVION!='Avion1' AND
(TO_DATE(HORA_SALIDA))!=TO_DATE('18-11-2016
3:05:08','DD-MM-YYYY hh:mi:ss') AND
(TO DATE(HORA LLEGADA))!=TO DATE('18-11-2016 3:05:08','DD-MM-YYYY hh:mi:ss'));
```

Distribución de los datos:

No hay un cambio significativo de la distribución con respecto a RF7, ya que se usan las mismas tablas, solo que el criterio de búsqueda es inverso. En el único que afecta es en la hora de llegada y salida, ya que ahora tiene que buscar entre todas las horas que no sean la seleccionada, teniéndose así una distribución muy grande de los datos para este caso. (Al rededor de 90%).

Se observó que si se cambia el rango de fechas a un rango más grande, el tiempo de ejecución es significativamente menor, ya que hay menos fechas de las que se pueden escoger los vuelos.



Análisis:

Oracle tiene un uso bastante ineficiente de los índices ya que se está haciendo un query de not equals, donde la selectividad es demasiado baja; por ello, se ven estos resultados tan elevados en costo. Sin embargo, se utiliza el índice de la llave primaria de viaje para poder hacer un range scan y no tener que recorrer toda la tabla al hacer los joins y las comparaciones.

RFC9:

SQL: consulta para encontrar la distancia recorrida por cada viajero y de sus viajes.
SELECT * FROM(SELECT * FROM((SELECT t1.ID, t1.TIPO_ID, SUM(t2.DURACION)as
millas FROM
((SELECT * FROM VIAJEROS JOIN (SELECT * FROM M_VIAJES_M_VIAJEROS) ON
VIAJEROS.ID = ID_VIAJEROS AND VIAJEROS.TIPO_ID = TIPO_ID_VIAJEROS)t1

```
JOIN (SELECT * FROM VUELO)t2 ON t1.ID_VUELO = t2.ID) GROUP BY t1.ID,
T1.TIPO_ID)t3 LEFT OUTER JOIN (SELECT * FROM M_VIAJES_M_VIAJEROS)t4
ON t3.ID = t4.ID_VIAJEROS AND t3.TIPO_ID = t4.TIPO_ID_VIAJEROS) WHERE MILLAS >
10)t4 JOIN (SELECT * FROM VIAJE_VIAJEROS)t5 ON t4.ID_VUELO =
t5.ID AND t4.HORA_SALIDA_VIAJE = t5.HORA_SALIDA;
```

Distribución de los datos:

Millas Viajadas: este nuevo parámetro puede hacer variar la selectividad de la consulta resultante de manera muy amplia debido a que en la mayoría de los casos los clientes habrán realizado un solo viaje y puede ser que la diferencia entre una milla haga que la selectividad cambie de gran manera. Hay selectividad alta.

Entre mayor sean las millas máximas, menos datos va a haber por lo que habrá menos complejidad.

| OPERATION | OBJECT_NAME | CARDINALITY | COST | |
|--------------------------------|---------------------|-------------|---------|--------|
| SELECT STATEMENT | | | 731215 | 103363 |
| HASH JOIN | | | 731215 | 103363 |
| Access Predicates | | | | |
| AND | | | | |
| T3.ID=M_VIAJES_M_VIAJEROS.ID_V | | | | |
| T3.TIPO_ID=M_VIAJES_M_VIAJEROS | | | | |
| VIEW | | | 1383340 | 30247 |
| FILTER | | | | |
| Filter Predicates | | | | |
| SUM(VUELO.DURACION)>10 | | | | |
| HASH (GROUP BY) | | | 1383340 | 30247 |
| NESTED LOOPS | | | 1383340 | 30214 |
| HASH JOIN | | | 1965067 | 30120 |
| Access Predicates | | | | |
| M_VIAJES_M_VIAJER | | | | |
| TABLE ACCESS (FULL) | VUELO | | 72476 | 272 |
| TABLE ACCESS (FULL) | M_VIAJES_M_VIAJEROS | | 1965062 | 3024 |
| INDEX (UNIQUE SCAN) | VIAJEROS_PK | 1 | | 0 |
| Access Predicates | | | | |
| AND | | | | |
| VIAJEROS.ID=M | | | | |
| VIAJEROS.TIPO_ | | | | |
| HASH JOIN | | | 1038705 | 39797 |
| Access Predicates | | | | |
| AND | | | | |
| M_VIAJES_M_VIAJEROS.ID_VUEL | | | | |
| M_VIAJES_M_VIAJEROS.HORA_S | | | | |
| TABLE ACCESS (FULL) | VIAJE_VIAJEROS | | 1038705 | 995 |
| TABLE ACCESS (FULL) | M_VIAJES_M_VIAJEROS | | 1965062 | 3025 |
| Other XML | | | | |

Para realizar este cálculo, Oracle optimiza el proceso de los joins mediante el índice primario de la llave primaria de los viajeros, que se compone del id y del tipo de id. Por ello, no tiene que recorrer toda la lista para encontrar las igualdades para hacer join.

RFC10:

SQL: Se busca el aeropuerto de las dos ciudades donde se quiere reportar sus viajes.

```
SELECT * FROM (select * FROM VIAJE WHERE TO_DATE(HORA_SALIDA) >=
TO_DATE('1-11-2016','DD-MM-YYYY') AND TO_DATE(HORA_LLEGADA) <=
TO_DATE('13-5-2017','DD-MM-YYYY'))t6
JOIN (SELECT * FROM (SELECT * FROM VUELO JOIN
(SELECT COD_IATA FROM AEROPUERTO WHERE NOMBRE_CIUADAD = 'Caracas')t2
```

```

ON VUELO.AEROPUERTO_SALIDA = t2.COD_IATA)t4 JOIN
(SELECT COD_IATA FROM AEROPUERTO WHERE NOMBRE_CIUADAD = 'New York')t3
ON
t3.COD_IATA = t4.AEROPUERTO_LLEGADA)t5 ON t5.ID =
t6.ID_VUELO;

```

Distribución de los datos:

Viajes entre aeropuertos: a pesar de que se intentó crear los datos de la forma más coherente posible no se puede garantizar una selectividad exacta al usar estos parámetros, ya que no se pudo asegurar que para un vuelo entrara mucha gente y que también hubiera muchas personas que hacen vuelos múltiples. (habría que crear muchos más usuarios).

SQL | 0.124 seconds

| OPERATION | OBJECT_NAME | CARDINALITY | COST | |
|-------------------------------|-------------|-------------|------|--|
| SELECT STATEMENT | | 1668 | 1052 | |
| NESTED LOOPS | | 1668 | 1052 | |
| NESTED LOOPS | | 1668 | 1052 | |
| HASH JOIN | | 464 | 279 | |
| Access Predicates | | | | |
| COD_IATA=VUELO.AEROPUERTO | | | | |
| TABLE ACCESS (FULL) | AEROPUERTO | 4 | 3 | |
| Filter Predicates | | | | |
| NOMBRE_CIUADAD='New York | | | | |
| HASH JOIN | | 5798 | 276 | |
| Access Predicates | | | | |
| VUELO.AEROPUERTO_SALIDA | | | | |
| TABLE ACCESS (FULL) | AEROPUERTO | 4 | 3 | |
| Filter Predicates | | | | |
| NOMBRE_CIUADAD='Cara | | | | |
| TABLE ACCESS (FULL) | VUELO | 72476 | 273 | |
| INDEX (RANGE SCAN) | VIAJE_PK | 1 | 1 | |
| Access Predicates | | | | |
| VUELO.ID=VIAJE.ID_VUELO | | | | |
| Filter Predicates | | | | |
| TO_DATE(INTERNAL_FUNCTION(| | | | |
| TABLE ACCESS (BY INDEX ROWID) | VIAJE | 4 | 2 | |
| Filter Predicates | | | | |
| TO_DATE(INTERNAL_FUNCTION(HOF | | | | |

Análisis:

Oracle busca eficiencia de los joins mediante el uso de nested loops y hash joins, ya que la información es coherente y ordenada. Además, se utilizan las llaves primarias para no tener que recorrer por completo las tablas.

Análisis del proceso de optimización:

Al pensar en la ejecución de un aplicación y su necesidad de traer los datos de la base a memoria principal, dependiendo de la granularidad de los datos, puede resultar más costoso que traer todos los datos que solo traer los datos procesados. Además, por las características de la base de datos y su implementación de consultas de bajo nivel, la implementación de procesos, especialmente aquellos que utilizan funciones nativas del sistema, resultaría mucho más costoso si se ejecutan en la aplicación.

Otro factor a pensar, es la magnitud de los datos y el tiempo que requiere procesarlos, ya que si la aplicación utiliza comandos de control y la magnitud está cerca a los millones o más, tendrían que utilizarse ciclos dobles o triples para ser capaces de recorrer toda la

información. Por lo tanto, la complejidad de las órdenes serán cuadráticas o mayores, causando un tiempo de ejecución muy alto. Este problema no sería un inconveniente en una solución nativa, aunque si las consultas son muy complejas sería ideal pensar en dividir las operaciones entre el programa y la base de datos.