Entrega Iteración 4: RotondAndes

Mauricio Neira, Juan Felipe Ramos

I. Documentación Diseño físico

Para la selección de índices procedemos a realizar un análisis por cada requerimiento funcional.

- Requerimiento Funcional 9

  Notemos que en la segunda tabla del JOIN, hay un WHERE que tiene como parámetro a $PRODUCTOS.RESTAURANTES\_ID$. Un índice secundario B+ sobre este parámetro ayudaría a reducir el tiempo de consulta considerablemente.

  Por otro lado, sería muy útil tener un arbol B+ asociado a la fecha de la tabla pues se está consultando un rango sobre este parámetro.

  Además, para hacer un HASH JOIN sobre los parámetros del JOIN, sería util crear un índice sobre $HISTORIAL.ID\_USUARIOS\_REGISTRADOS$.

  Ahora bien, veamos los índices creados por defecto por ORACLE en las tablas $USUARIOS\_REGISTRADOS$, $PRODUCTOS$ e $HISTORIAL$:

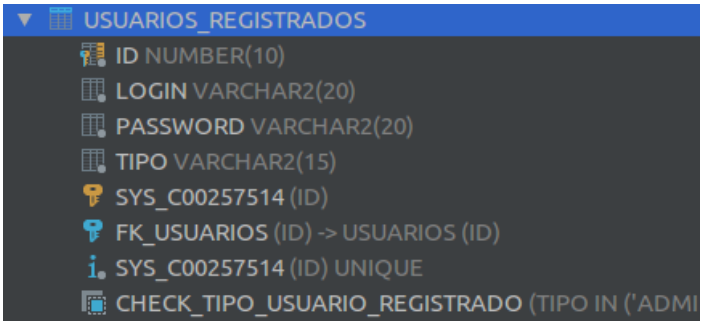

Figura 1: Indices por defecto de la tabla $USUARIOS\_REGISTRADOS$.

  Como ID es la llave primaria, ORACLE hace un índice sobre este atributo por defecto para optimizar las consultas que usan el PK.

M. Neira 201424001, m.neira10@uniandes.edu.co
J.F. Ramos 201616932, jf.ramos@uniandes.edu.co

Figura 2: Indices por defecto de la tabla $PRODUCTOS$.

  Como ID es la llave primaria, ORACLE hace un índice sobre este atributo por defecto para optimizar las consultas que usan el PK.

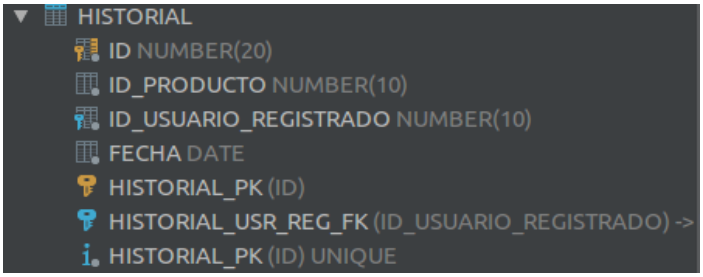

Figura 3: Indices por defecto de la tabla $HISTORIAL$.

  Como ID es la llave primaria, ORACLE hace un índice sobre este atributo por defecto para optimizar las consultas que usan el PK.

- Requerimiento Funcional 10

  Este requerimiento es igual a RFC9 salvo que se resta el conjunto de la tabla $USUARIOS\_REGISTRADOS$. Por esta razón, todo lo que se dijo en el punto anterior aplica para este. No hay creación de indices adicional que ayude a hacer la consulta mas eficiente pues el MINUS se hace sobre la PK de $USUARIOS\_REGISTRADOS$ y éste parámetro tiene por defecto un índice asociado.

- Requerimiento Funcional 11

  Este requerimiento esta partido en 4:

- Consultar el producto mas consumido

- Consultar el producto menos consumido

- Consultar el restaurante mas frecuentado

- Consultar el restaurante menos frecuentado

Para los 4 casos se observa que existe un JOIN sobre la función $TO\_CHAR(HISTORIAL.FECHA,'d')$ y $HISTORIAL.ID_PRODUCTO$. Para hacer la consulta más eficiente, se podría facilitar un HASH JOIN. Así pues, se harían índices secundarios sobre los dos parámetros descritos anteriormente en forma de hash. Nótese que el primer índice es funcional.

En estos requerimientos, las tablas utilizadas son $HISTORIAL$ y $PRODUCTOS$:

Ambas tablas y sus indices se discutieron en RFC9.

- Requerimiento Funcional 12

Para el requerimiento 12 tenemos que se realizan varios join en diferentes criterios por lo cual sería conveniente que se hagan índices, en particular tanto sobre $ID\_PRODUCTO$ como sobre $ID\_USUARIO\_REGISTRADO$. Cabe resaltar que en este caso no se debe hacer un árbol B+ sobre los precios dado el gran rango de búsqueda (en este caso 50 por ciento). Dada la sentencia, es necesario crear un índice sobre el conjunto de funciones usadas para la sentencia. De esa manera, es ideal tener un índice sobre la función $TO\_NUMBER(TO\_CHAR(HISTORIAL.FECHA,'WW')) - 1$ para realizar las posibles consultas sobre esos valores de la manera más eficiente. En particular, dado que no buscamos rangos de esos valores, sería útil una tabla de hash. Para finalizar, es pertinente señalar que la selectividad de dicha función es 7 sobre el numero de datos, que aunque baja es lo suficientemente pertinente como para considerarla en el proceso de optimización. Señalamos finalmente que los resultados de esta función están bien distribuidos gracias a que la entrada de datos está bien distribuida.
En cuanto a la creación por defecto de Oracle de los índices de las tablas propuestas, estos indices son los mismos de HISTORIAL y PRODUCTOS

II. Documentación RFC 9

II-A. *Documentación Escenario de Prueba*

- Sentencia SQL

```
SELECT *
FROM
  USUARIOS_REGISTRADOS
  JOIN (SELECT ID_USUARIO_REGISTRADO
        FROM PRODUCTOS JOIN HISTORIAL ON
            (HISTORIAL.ID_PRODUCTO =
            PRODUCTOS.ID)
        WHERE PRODUCTOS.RESTAURANTES_ID =
            21 AND
            (to_date(to_char(HISTORIAL.FECHA,
            'YYYY/MM/DD HH24:MI:SS'),
            'YYYY/MM/DD HH24:MI:SS')
            between to_date('2017/01/01
            00:00:00', 'YYYY/MM/DD
            HH24:MI:SS') and
            to_date('2017/05/01
            00:00:00', 'YYYY/MM/DD
            HH24:MI:SS')))
  ON (USUARIOS_REGISTRADOS.ID =
      ID_USUARIO_REGISTRADO);
```

- Distribución de datos

Cuando el rango de las fechas es grande, se espera que el tamaño de respuesta aumente considerablemente pues aumenta la cantidad de tuplas en HISTORIAL que cumplen estar en el rango de fechas.

- Parámetros del análisis

$$fecha1 =' 2017/01/0100:00:00'$$
$$fecha2 =' 2017/05/0100:00:00'$$

- Plan de consulta

Figura 4: Plan de consulta para RFC9.

II-B. *Análisis de eficiencia*

- Escenarios posibles
- Plan de ejecución propuesto
- Diferencia entre plan Oracle y plan propuesto

III. Documentación RFC 10

III-A. *Documentación Escenario de Prueba*

- Sentencia SQL

```
SELECT * FROM USUARIOS_REGISTRADOS
MINUS
(SELECT USUARIOS_REGISTRADOS.*
FROM
  USUARIOS_REGISTRADOS
  JOIN (SELECT ID_USUARIO_REGISTRADO
        FROM PRODUCTOS JOIN HISTORIAL ON
            (HISTORIAL.ID_PRODUCTO =
            PRODUCTOS.ID)
```

```
        WHERE PRODUCTOS.RESTAURANTES_ID =
            21 AND
            (to_date(to_char(HISTORIAL.FECHA,
            'YYYY/MM/DD HH24:MI:SS'),
            'YYYY/MM/DD HH24:MI:SS')
            between to_date('2017/01/01
            00:00:00', 'YYYY/MM/DD
            HH24:MI:SS') and
            to_date('2017/05/01
            00:00:00', 'YYYY/MM/DD
            HH24:MI:SS')))
    ON (USUARIOS_REGISTRADOS.ID =
        ID_USUARIO_REGISTRADO));
```

- Distribución de datos

  Aquí pasa lo opuesto al requerimiento anterior. Al aumentar el rango de fechas, al hacer el MINUS, la cantidad de tuplas resultantes disminuye. Así pues, el tamaño de respuesta disminuye al incrementar el rango de fechas.

- Parámetros del análisis

$$fecha1 =' 2017/01/0100:00:00'$$
$$fecha2 =' 2017/05/0100:00:00'$$

- Plan de consulta

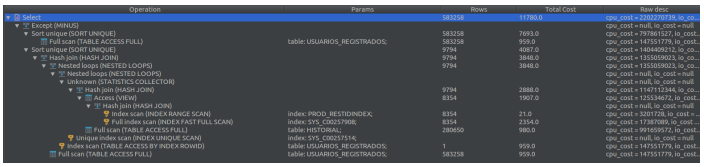

Figura 5: Plan de consulta para RFC10.

III-B. Análisis de eficiencia

- Escenarios posibles
- Plan de ejecución propuesto
- Diferencia entre plan Oracle y plan propuesto

IV. Documentación RFC 11

IV-A. Documentación Escenario de Prueba

- Sentencias SQL
  - Producto más consumido

```
WITH TEMP AS (
     SELECT DIA1, MAX(CONT) CONTEO
       FROM
          (SELECT ID_PRODUCTO, DIA AS
              DIA1, CONT
            FROM
              (SELECT H.ID_PRODUCTO
                  ID_PRODUCTO,
                  to_char(H.FECHA,
                  'd') DIA ,
                  COUNT(H.ID_PRODUCTO)
                  CONT
                    FROM HISTORIAL H
                    GROUP BY
                        to_char(H.FECHA,
                        'd'),
                        H.ID_PRODUCTO))
          GROUP BY DIA1) ,

    TEMP2 AS (
      SELECT *
        FROM TEMP
        JOIN (SELECT ID_PRODUCTO, DIA
            AS DIA2, CONT
              FROM (SELECT
                  H.ID_PRODUCTO
                  ID_PRODUCTO,
                  to_char(H.FECHA, 'd')
                  DIA ,
                  COUNT(H.ID_PRODUCTO)
                  CONT
                    FROM HISTORIAL H
                    GROUP BY
                        to_char(H.FECHA,
                        'd'),
                        H.ID_PRODUCTO))
          ON TEMP.DIA1 = DIA2 AND
              TEMP.CONTEO = CONT)

SELECT DISTINCT DIA2 AS
    DIA_SEMANA,CONT AS CANTIDAD ,
    first_value(ID_PRODUCTO)
  OVER (PARTITION BY DIA2,CONT ) AS
      PRODUCTO_ID
  FROM TEMP2
  ORDER BY DIA2 ASC;
```

- Producto menos consumido

```
WITH TEMP AS (
     SELECT DIA1, MIN(CONT) CONTEO
        FROM (SELECT ID_PRODUCTO,
        DIA AS DIA1, CONT
       FROM (SELECT H.ID_PRODUCTO
           ID_PRODUCTO,
           to_char(H.FECHA, 'd') DIA ,
           COUNT(H.ID_PRODUCTO) CONT
             FROM HISTORIAL H
             GROUP BY
                 to_char(H.FECHA,
                 'd'), H.ID_PRODUCTO))
      GROUP BY DIA1) ,
    TEMP2 AS (
      SELECT *
        FROM TEMP
        JOIN (SELECT ID_PRODUCTO, DIA
            AS DIA2, CONT
        FROM (SELECT H.ID_PRODUCTO
            ID_PRODUCTO,
            to_char(H.FECHA, 'd') DIA ,
            COUNT(H.ID_PRODUCTO) CONT
              FROM HISTORIAL H
              GROUP BY
                  to_char(H.FECHA,
                  'd'), H.ID_PRODUCTO))
```

```
        ON TEMP.DIA1 = DIA2 AND
            TEMP.CONTEO = CONT)
SELECT DISTINCT DIA2 AS DIA_SEMANA ,
    CONT AS CANTIDAD ,
    first_value(ID_PRODUCTO)
  OVER (PARTITION BY DIA2,CONT ) AS
      PRODUCTO_ID
  FROM TEMP2
  ORDER BY DIA2 ASC;
```

- Restaurante más frecuentado

```
WITH TEMP AS (
SELECT DIA1, MAX(CONT) CONTEO
FROM (SELECT RESTAURANTES_ID, DIA AS
    DIA1, CONT
   FROM (SELECT RESTAURANTES_ID,
       to_char(H.FECHA, 'd') DIA ,
       COUNT(P.RESTAURANTES_ID) CONT
             FROM (HISTORIAL H JOIN
                 PRODUCTOS P
                 ON(H.ID_PRODUCTO=P.ID))
     GROUP BY to_char(H.FECHA, 'd'),
         P.RESTAURANTES_ID))
       GROUP BY DIA1) ,
TEMP2 AS (
SELECT *
FROM TEMP
 JOIN (SELECT RESTAURANTES_ID, DIA AS
     DIA2, CONT
     FROM (SELECT RESTAURANTES_ID,
         to_char(H.FECHA, 'd') DIA ,
         COUNT(P.RESTAURANTES_ID) CONT
       FROM (HISTORIAL H JOIN
           PRODUCTOS P
           ON(H.ID_PRODUCTO=P.ID))
       GROUP BY to_char(H.FECHA,
           'd'), P.RESTAURANTES_ID))
   ON TEMP.DIA1 = DIA2 AND
       TEMP.CONTEO = CONT)
SELECT DISTINCT DIA2 AS
    DIA_SEMANA,CONT AS CANTIDAD,
    first_value(RESTAURANTES_ID)
 OVER (PARTITION BY DIA2,CONT) AS
     ID_RESTAURANTE
FROM TEMP2
ORDER BY DIA2 ASC;
```

- Restaurante menos frecuentado

```
WITH TEMP AS (
SELECT DIA1, MIN(CONT) CONTEO
FROM (SELECT RESTAURANTES_ID, DIA AS DIA1, CONT
   FROM (SELECT RESTAURANTES_ID, to_char(H.FECHA, 'd') DIA ,
           COUNT(P.RESTAURANTES_ID) CONT
             FROM (HISTORIAL H JOIN PRODUCTOS P ON(H.ID_PRODUCTO=P.ID))
      GROUP BY to_char(H.FECHA, 'd'), P.RESTAURANTES_ID))
       GROUP BY DIA1) ,
TEMP2 AS (
SELECT *
FROM TEMP
 JOIN (SELECT RESTAURANTES_ID, DIA AS DIA2, CONT
      FROM (SELECT RESTAURANTES_ID, to_char(H.FECHA, 'd') DIA ,
              COUNT(P.RESTAURANTES_ID) CONT
        FROM (HISTORIAL H JOIN PRODUCTOS P ON(H.ID_PRODUCTO=P.ID))
        GROUP BY to_char(H.FECHA, 'd'), P.RESTAURANTES_ID))
   ON TEMP.DIA1 = DIA2 AND TEMP.CONTEO = CONT)
SELECT DISTINCT DIA2 AS DIA_SEMANA,CONT AS CANTIDAD,
         first_value(RESTAURANTES_ID)
 OVER (PARTITION BY DIA2,CONT) AS ID_RESTAURANTE
FROM TEMP2
ORDER BY DIA2 ASC;
```

- Distribución de datos

  La distribucion de los datos no se ven afectados por los parametros de entrada pues no hay parametros de entrada.

- Parámetros del análisis

  No hay parámetros de entrada.
- Plan de consulta

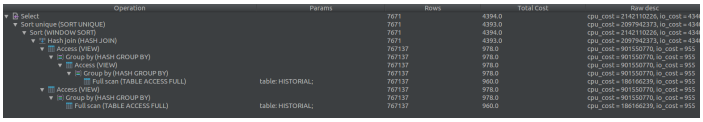

Figura 6: Plan de consulta para RFC11 mas consumido.

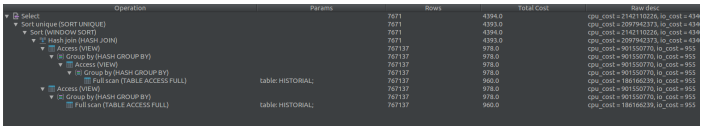

Figura 7: Plan de consulta para RFC11 menos consumido.

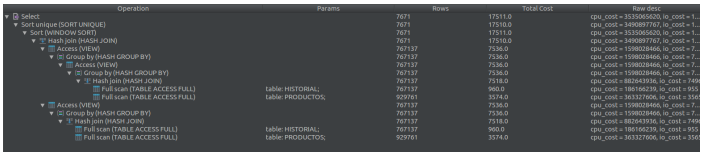

Figura 8: Plan de consulta para RFC11 mas frecuentado.

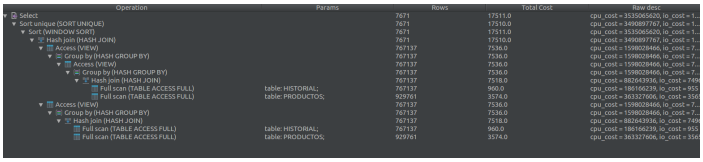

Figura 9: Plan de consulta para RFC11 menos frecuentadoo.

IV-B. Análisis de eficiencia

- Escenarios posibles
- Plan de ejecución propuesto
- Diferencia entre plan Oracle y plan propuesto

V. Documentación RFC 12

V-A. Documentación Escenario de Prueba

- Sentencia SQL

```
(SELECT *
FROM USUARIOS_REGISTRADOS
  NATURAL JOIN
  (SELECT ID_USUARIO_REGISTRADO AS ID
      FROM HISTORIAL
      GROUP BY ID_USUARIO_REGISTRADO
      HAVING (COUNT(DISTINCT TO_NUMBER(TO_CHAR(HISTORIAL.FECHA, 'WW')))-1
          = TO_NUMBER(TO_CHAR(SYSDATE, 'WW')) -
                MIN(TO_NUMBER(TO_CHAR(HISTORIAL.FECHA, 'WW')) ) ) ))

UNION

(SELECT *
FROM USUARIOS_REGISTRADOS WHERE TIPO LIKE 'USUARIO'
MINUS
(SELECT U.*
FROM USUARIOS_REGISTRADOS U JOIN (HISTORIAL H JOIN MENUS M ON
          (H.ID_PRODUCTO=M.ID)) ON (U.ID=ID_USUARIO_REGISTRADO)))

UNION

(SELECT *
FROM USUARIOS_REGISTRADOS WHERE TIPO LIKE 'USUARIO'

MINUS

(SELECT U.*
FROM USUARIOS_REGISTRADOS U JOIN (HISTORIAL H1 JOIN PRODUCTOS P ON(
          H1.ID_PRODUCTO = P.ID AND P.PRECIO <= 36885.84)) ON
          (U.ID=ID_USUARIO_REGISTRADO)));
```

- Distribución de datos

  La distribución de los datos no se ven afectados por los parámetros de entrada pues no hay parámetros de entrada.

- Parámetros del análisis

  No hay parámetros de entrada.

- Plan de consulta

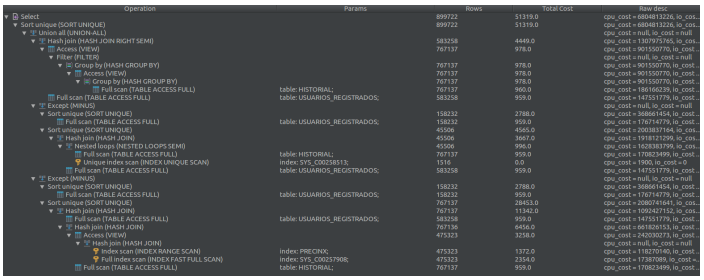

Figura 10: Plan de consulta para RFC12.

V-B. Análisis de eficiencia

- Escenarios posibles
- Plan de ejecución propuesto
- Diferencia entre plan Oracle y plan propuesto

VI. Documentación Proceso de carga de datos

Se usaron scripts en python con números aleatorios y csv's externos para generar archivos csv que fueron importados a través de Datagrip a la base de datos.

Por ejemplo, para la creación de los usuarios, se descargó de varios repositorios de github csv's con la información de los nombres y apellidos de hombres y mujeres en latinoamerica. Se generaron números aleatorios para escoger nombres al azar. El script completo se puede ver a continuación:

```
import numpy as np
import pandas as pd
from StringIO import StringIO


numDatos=1000000
datos = []


hombres = pd.read_csv('hombres.csv', delimiter=',')
mujeres = pd.read_csv('mujeres.csv', delimiter=',')
 apellidos = pd.read_csv(' apellidos .csv', delimiter=',')
lenh = (len(hombres.nombre))
lenm = (len(mujeres.nombre))
lena = (len( apellidos . apellido ))

for i in range(numDatos):
    temp = []
    temp.append(i+31)
    genero = np.random.random()
    if (genero<0.5):
        temp.append(hombres.iloc[ int (np.random.random()*lenh//1)]['nombre'])
    else :
        temp.append(mujeres. iloc [ int (np.random.random()*lenm//1)]['nombre'])
    temp.append( apellidos . iloc [ int (np.random.random()*lena//1)][' apellido '])
    temp.append("CC")
    temp.append("1000000000"+str(i+31))
    temp.append("email"+str(i+31)+"@whatev.com")
    datos.append(temp)
    print ((i+0.0)*100/numDatos)


datos = np.array(datos)


fmt=' %s, %s, %s, %s, %s, %s'
np.savetxt("usuarios .csv", datos, delimiter=",",fmt = fmt)
```

Todos los scripts se encuentran en "/Docs/populate/" si se requiere consultarlos.

Es importante resaltar que se poblaron las tablas con 10^6 tuplas donde tenía sentido poblarlas con esa cantidad de datos. Tablas como ZONAS y RESTAURANTES no tienen esa cantidad de datos pues no tiene sentido dentro del modelo.

VII. Análisis proceso de optimización

La base de datos está hecha para optmizar todas las consultas que se le hacen. Incluyendo a las sentencias mismas. Implementar soluciones desde la capa lógica es una manera ingenua de resolver un problema que ya ha sido solucionado de manera eficiente. Más aún cuando los datos de la consulta no caben en memoria.