

# 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 árbol 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 útil 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*:

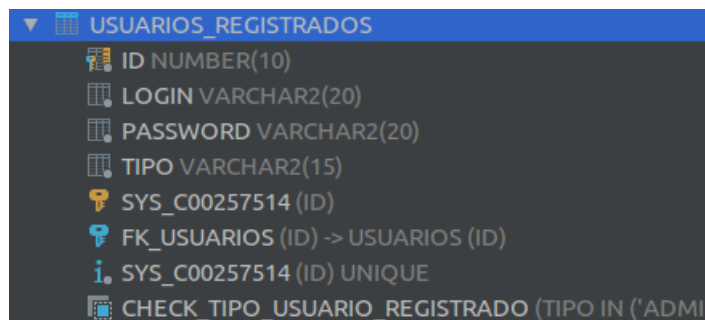


Figura 1: Índices por defecto de la tabla *USUARIOS\_REGISTRADOS*.

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



Figura 2: Índices 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.

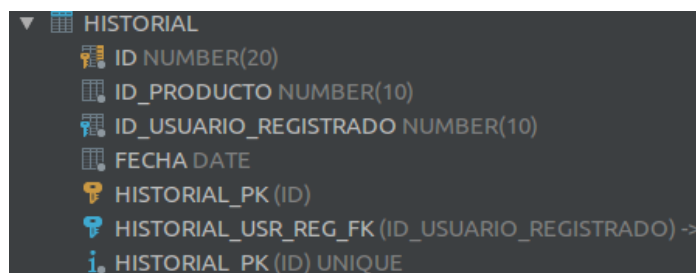


Figura 3: Índices por defecto de la tabla *HISTORIAL*.

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 índices 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 índices 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 índices 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

Operation	Params	9794	Rows	Total Cost	Row Cost
Hash join (HASH JOIN)		9794	3848.0	155509623.0	155509623.0
Nested loops (NESTED LOOPS)		9794	3848.0	155509623.0	155509623.0
Nested loops (NESTED LOOPS)		9794	3848.0	155509623.0	155509623.0
Access (VIEW)		9794	3848.0	155509623.0	155509623.0
Hash join (HASH JOIN)		9794	3848.0	155509623.0	155509623.0
Index scan (INDEX RANGE SCAN)	INDEX: PRDCTO_RESTAURANTES	9794	3848.0	155509623.0	155509623.0
Full index scan (INDEX FAST FULL SCAN)	INDEX: SYS_C00237908	9794	3848.0	155509623.0	155509623.0
Full table scan (TABLE ACCESS FULL)	TABLE: HISTORIAL	9794	3848.0	155509623.0	155509623.0
Unique index scan (INDEX UNIQUE SCAN)	INDEX: SYS_C00237116	9794	3848.0	155509623.0	155509623.0
Index scan (TABLE ACCESS BY INDEX ROWID)	TABLE: USUARIOS_REGISTRADOS	9794	3848.0	155509623.0	155509623.0
Full table scan (TABLE ACCESS FULL)	TABLE: USUARIOS_REGISTRADOS	9794	3848.0	155509623.0	155509623.0

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

Plan	Operation	Params	Rows	Total Cost	Estimated Rows
1	Sort (TEMP SORT)		582128	11789.0	582128
2	Sort output (SORT UNIQUE)		582128	7693.0	582128
3	Full scan (TABLE ACCESS FULL)	TABLE: LUNARLOS, REGISTRATION	999.0	115519.0	999.0
4	Sort output (SORT UNIQUE)		999.0	1084.0	999.0
5	Full scan (TABLE ACCESS FULL)	TABLE: LUNARLOS, REGISTRATION	346.0	115519.0	346.0
6	Nested loops (NESTED LOOPS)		9794	131550.0	9794.0
7	Nested loops (NESTED LOOPS)		9794	131550.0	9794.0
8	UNION (UNION)		9794	131550.0	9794.0
9	UNION (UNION)		9794	131550.0	9794.0
10	UNION (UNION)		9794	131550.0	9794.0
11	UNION (UNION)		9794	131550.0	9794.0
12	UNION (UNION)		9794	131550.0	9794.0
13	UNION (UNION)		9794	131550.0	9794.0
14	UNION (UNION)		9794	131550.0	9794.0
15	UNION (UNION)		9794	131550.0	9794.0
16	UNION (UNION)		9794	131550.0	9794.0
17	UNION (UNION)		9794	131550.0	9794.0
18	UNION (UNION)		9794	131550.0	9794.0
19	UNION (UNION)		9794	131550.0	9794.0
20	UNION (UNION)		9794	131550.0	9794.0
21	UNION (UNION)		9794	131550.0	9794.0
22	UNION (UNION)		9794	131550.0	9794.0
23	UNION (UNION)		9794	131550.0	9794.0
24	UNION (UNION)		9794	131550.0	9794.0
25	UNION (UNION)		9794	131550.0	9794.0
26	UNION (UNION)		9794	131550.0	9794.0
27	UNION (UNION)		9794	131550.0	9794.0
28	UNION (UNION)		9794	131550.0	9794.0
29	UNION (UNION)		9794	131550.0	9794.0
30	UNION (UNION)		9794	131550.0	9794.0
31	UNION (UNION)		9794	131550.0	9794.0
32	UNION (UNION)		9794	131550.0	9794.0
33	UNION (UNION)		9794	131550.0	9794.0
34	UNION (UNION)		9794	131550.0	9794.0
35	UNION (UNION)		9794	131550.0	9794.0
36	UNION (UNION)		9794	131550.0	9794.0
37	UNION (UNION)		9794	131550.0	9794.0
38	UNION (UNION)		9794	131550.0	9794.0
39	UNION (UNION)		9794	131550.0	9794.0
40	UNION (UNION)		9794	131550.0	9794.0
41	UNION (UNION)		9794	131550.0	9794.0
42	UNION (UNION)		9794	131550.0	9794.0
43	UNION (UNION)		9794	131550.0	9794.0
44	UNION (UNION)		9794	131550.0	9794.0
45	UNION (UNION)		9794	131550.0	9794.0
46	UNION (UNION)		9794	131550.0	9794.0
47	UNION (UNION)		9794	131550.0	9794.0
48	UNION (UNION)		9794	131550.0	9794.0
49	UNION (UNION)		9794	131550.0	9794.0
50	UNION (UNION)		9794	131550.0	9794.0
51	UNION (UNION)		9794	131550.0	9794.0
52	UNION (UNION)		9794	131550.0	9794.0
53	UNION (UNION)		9794	131550.0	9794.0
54	UNION (UNION)		9794	131550.0	9794.0
55	UNION (UNION)		9794	131550.0	9794.0
56	UNION (UNION)		9794	131550.0	9794.0
57	UNION (UNION)		9794	131550.0	9794.0
58	UNION (UNION)		9794	131550.0	9794.0
59	UNION (UNION)		9794	131550.0	9794.0
60	UNION (UNION)		9794	131550.0	9794.0
61	UNION (UNION)		9794	131550.0	9794.0
62	UNION (UNION)		9794	131550.0	9794.0
63	UNION (UNION)		9794	131550.0	9794.0
64	UNION (UNION)		9794	131550.0	9794.0
65	UNION (UNION)		9794	131550.0	9794.0
66	UNION (UNION)		9794	131550.0	9794.0
67	UNION (UNION)		9794	131550.0	9794.0

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)

LECT 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 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 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 frecuentado.

### 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_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 H JOIN PRODUCTOS P ON(
H.ID_PRODUCTO = P.ID AND P.PRECIO <= 36855.84)) 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

Operation	Params	Rows	Total Cost	Rows/Doc
Sort Unique (SORT UNIQUE)		899722	5139.0	899722
Union all (UNION ALL)		899722	5139.0	899722
Hash Join (HASH JOIN)		582356	4446.0	582356
Access (VIEW)		767137	978.0	767137
Hash Join (HASH JOIN)		767137	978.0	767137
Access (VIEW)		767137	978.0	767137
Hash Join (HASH JOIN)		767137	978.0	767137
Access (VIEW)		767137	978.0	767137
Full Scan (TABLE ACCESS FULL)	TABLE HISTORIAL	767137	978.0	767137
Full Scan (TABLE ACCESS FULL)	TABLE USUARIOS_REGISTRADOS	582356	959.0	582356
Sort Unique (SORT UNIQUE)	TABLE USUARIOS_REGISTRADOS	156232	2788.0	156232
Full Scan (TABLE ACCESS FULL)	TABLE USUARIOS_REGISTRADOS	156232	959.0	156232
Sort Unique (SORT UNIQUE)	TABLE USUARIOS_REGISTRADOS	45506	4565.0	45506
Hash Join (HASH JOIN)	TABLE HISTORIAL	45506	965.0	45506
Full Scan (TABLE ACCESS FULL)	TABLE USUARIOS_REGISTRADOS	45506	965.0	45506
Hash Join (HASH JOIN)	TABLE HISTORIAL	767137	978.0	767137
Full Scan (TABLE ACCESS FULL)	TABLE USUARIOS_REGISTRADOS	1516	0.0	1516
Hash Join (HASH JOIN)	TABLE HISTORIAL	582356	959.0	582356
Sort Unique (SORT UNIQUE)	TABLE USUARIOS_REGISTRADOS	156232	2788.0	156232
Full Scan (TABLE ACCESS FULL)	TABLE USUARIOS_REGISTRADOS	156232	959.0	156232
Sort Unique (SORT UNIQUE)	TABLE USUARIOS_REGISTRADOS	767137	11462.0	767137
Hash Join (HASH JOIN)	TABLE HISTORIAL	767137	978.0	767137
Full Scan (TABLE ACCESS FULL)	TABLE USUARIOS_REGISTRADOS	582356	959.0	582356
Hash Join (HASH JOIN)	TABLE HISTORIAL	767137	978.0	767137
Full Scan (TABLE ACCESS FULL)	TABLE USUARIOS_REGISTRADOS	475323	3258.0	475323
Hash Join (HASH JOIN)	TABLE HISTORIAL	475323	1372.0	475323
Full Scan (TABLE ACCESS FULL)	TABLE USUARIOS_REGISTRADOS	475323	3258.0	475323
Full Scan (TABLE ACCESS FULL)	TABLE HISTORIAL	767137	978.0	767137

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 optimizar 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.