Iteración 4

Zulma Lorena Castañeda Hidalgo, Andrés Felipe Pinzón Adame
Sistemas Transaccionales
Universidad de los Andes, Bogotá, Colombia
{zl.castaneda10, af.pinzon10}@uniandes.edu.co
Fecha de presentación: Mayo 7 de 2017

1. Análisis

Con respecto a la iteración anterior y con el objetivo de implementar los nuevos requerimientos se realizaron modificaciones en el nombre de algunas columnas de la base de datos.

Con el propósito de realizar natural joins fácilmente debido a la gran cantidad de tablas involucradas, debido al nivel de normalización de la base de datos, Modificamos el nombre de las columnas nombre de algunas tablas para diferenciarlas unas de otras y así permitir hacer natural join de la manera más simple.

Para esta iteración no se añadieron nuevas tablas pues las que ya tenemos cumplen con todos los requerimientos.

En cuanto al modelo del mundo sólo se creó un nuevo DAO para atender a estas consultas y se añadieron algunos servicios a la parte REST y métodos a la clase principal de FestivAndes.

2. Diseño de la aplicación

2.1 Impacto introducción requerimientos

Con respecto a las iteraciones anteriores, no existe un impacto directo sobre las consultas pues no sé añadieron nuevas tablas a la base de datos por esta razón no se analizarán cambios en esta sección.

2.2 Diseño físico

- Documente su diseño físico
- **Selección de índices para cada requerimiento funcional**

RFC9 Consultar asistencia a FestivAndes: En este requerimiento se pide consultar los clientes que asistieron a alguna función en el festival filtrado por IdCompañia, si la función fue realizada, fecha y un parámetro de ordenamiento que debe pertenecer a alguno de lo datos de cliente. Se analizó crear los siguientes índices.

- **Rango de fechas (Función.Fecha):** Se utiliza un índice B+ pues es la mejor opción para buscar rápidamente datos entre rangos, debido a su estructura natural de agrupar los datos. Las fechas en que ocurren los espectáculos se encuentran distribuidas de manera relativamente uniforme lo cual hace ideal el uso de este tipo de índice. Se usa sobre la columna Fecha de la tabla Función que indica la fecha en que se llevará o se llevó a cabo.

- **Realizado:** Para el campo Realizado de la tabla Función se optó por utilizar un índice BITMAP. Esto debido a que a pesar de que son pocas las opciones que este campo puede tomar (1 para realizado, 0 para por realizar y 2 para cancelado), la cantidad de registros que va a tener esta tabla amerita el uso de este índice. Además, como la información de la tabla Funciones no es muy dada a modificaciones (es inusual y solo ocurre con funciones por realizar que son un mínimo porcentaje) no existe el problema de constantes actualizaciones para el índice.

- **idCompania:** Para este campo se utiliza el índice por defecto que crea la base de datos pues es un campo de tipo llave primaria.

Finalmente para las columnas de ID mediante las cuales se realizan los join de obra y función además de compañía/obra y compañía se utiliza el índice por defecto de SQL (B+).

RFC10: Consultar asistencia a FestivAndes V2

Este requerimiento es muy similar al anterior, solo que lo opuesto pues se solicita la misma información de los clientes, pero esta vez que NO asistieron a ninguna función en ese rango de fechas, por lo tanto los mismos índices del requerimiento anterior aplican para esta consulta.

La diferencia principal a nivel de consulta radica en que se debe realizar la consulta anterior además de un acceso completo a la tabla de clientes para poder encontrar la diferencia (MINUS) entre ellas. Los clientes que no aparecieron en la consulta anterior deberán aparecer en esta.

RFC11 Consultar compra de boletas:
En esta consulta se pide consultar las boletas compradas según la función dando la opción de filtrar por varios campos, se analizará los índices planteados.

- **Rango de fecha:** Para este campo se realizó la implementación de un árbol B+ pues es el ideal para búsquedas de rangos como ya se explicó anteriormente.

- **Elementos del escenario:** Como el porcentaje de tuplas que poseen un elemento específico del escenario es mínima se optó por introducir un índice tipo BITMAP.

- **Tipo localidad:** Como los tipos de localidad de las boletas vendidas se encuentran distribuidas de manera relativamente uniforme se utilizará BITMAP, pues según el diseño de nuestra base de datos, si bien no hay muchos tipos de localidad, si hay muchas localidades diferentes para cada espacio lo cual incrementa el número de registros en esta tabla. Sin embargo, estas localidades no están sujetas a muchas modificaciones ni se añaden nuevas muy a menudo, lo cual hace que este índice valga la pena.
- **Franja horaria y dia de la semana:** Como el campo del cual conseguimos esta información se encuentra dentro de la columna fecha EXPERIMENTAR.

RFC12 Consultar los buenos clientes:

En esta consulta se pide consultar los clientes que han comprando mas de n boletas en VIP para ella se consideraron los siguientes índices.

- **Localidad:** Al igual que la consulta anterior, el índice indicado para este campo es BITMAP debido al tamaño de la tabla y la distribución uniforme de los tipos de localidad que existen. Como la consulta requiere acceder a esta información en su cláusula WHERE es un índice apropiado.

- **Conteo de boletas:** Como para contar cuántas boletas tiene cada persona es necesario realizar un recorrido completo de la tabla de boletas. Por esta razón un índice no ayudaría a mejorar la consulta.

■ Índices creados por Oracle

Ejecutamos la siguiente consulta para conocer los índices creados por ORACLE.

SELECT index_name, clustering_factor, index_type **FROM** user_indexes;

	INDEX_NAME	CLUSTERING_FACTOR	INDEX_TYPE
1	ABONO_PK	2	NORMAL
2	BEBEDORES_PK	24	NORMAL
3	BOLETASABONADAS_PK	0	NORMAL
4	BOLETA_PK	4	NORMAL
5	CATEGORIA_OBRA_PK	1	NORMAL
6	CATEGORIA_PK	1	NORMAL
7	CLIENTE_PK	1	NORMAL
8	COMPANIAOBRA_PK	1	NORMAL
9	COMPANIA_PK	1	NORMAL
10	DESCRIPCIONESESPACIO_PK	(null)	NORMAL
11	DESCRIPCIONTECNICA_PK	(null)	NORMAL
12	ESPACIO_PK	1	NORMAL
13	FUNCION_PK	1	NORMAL
14	LOCALIDAD_PK	1	NORMAL
15	OBRA_PK	1	NORMAL
16	PK_RESERVAS	(null)	NORMAL
17	PK_SILLASRESERVAS	(null)	NORMAL
18	PREFERENCIA_PK	1	NORMAL
19	PRODUCCION_PK	1	NORMAL
20	ROL_PK	1	NORMAL
21	USUARIO_PK	1	NORMAL

Todos los índices que Oracle crea por defecto son de tipo NORMAL, es decir son de árbol B+, únicos y primarios (*clustered*) por lo que solo se puede tener un índice de este tipo en cada tabla. Estos índices fueron creados por ORACLE pues corresponden a las llaves primarias de cada una de las tablas de la base de datos. Esto es debido a que la selectividad de la llave primaria de una tabla es muy alta y cuando se realiza una consulta con un WHERE sobre esta llave primaria, es mucho más rápido encontrarla si posee este índice.

Tener este tipo de índices en la llave primaria es útil pues ayuda al rendimiento general de los requerimientos funcionales (casi todos requieren el uso de llaves primarias en algún punto, ya sea a la hora de hacer joins o de encontrar una tupla específica)

Los índices particularmente útiles para los requerimientos funcionales de consulta de esta iteración son:

- **Funcion_PK, Cliente_PK y Compania_PK:** Permite la unión rápida entre estas tablas para todos los requerimientos, Se encuentran sobre la columna ID de cada tabla..
- **CompaniaObra_PK y DescripcionesEspacio_PK:** Permite la unión tablas principales ayudando a la normalización, estas llaves primarias tienen más de un campo y por lo tanto son combinadas. Se utilizan en el requerimientos RFC11 y solo CompaniaObra_PK en el RFC9 y RFC10.
- **Localidad_PK:** Pertinente a las uniones usadas en los requerimientos RFC11 y RFC12 para localizar rápidamente el tipo de localidad.

■ Análisis para cada requerimiento

RFC9:

■ Documentación de escenarios de prueba

- **Sentencia SQL:** Para este requerimiento se creo una consulta que une las tablas cliente, boleta, funcion, obra, companiaobra y compania; que trae los datos de los clientes dada una compañia, que hayan asistido a una función de dicha compañía entre un lapso de tiempo establecido, y se ordena por una característica definida

```
SELECT DISTINCT IDCLIENTE, T3.NOMBRE,APELLIDO, T3.EMAIL, IDENTIFICACION FROM
(SELECT IDCLIENTE, T1.NOMBRE, APELLIDO, EMAIL, IDENTIFICACION, IDOBRA FROM
(SELECT IDCLIENTE, NOMBRE, APELLIDO, EMAIL, IDENTIFICACION, IDFUNCION FROM CLIENTE NATURAL JOIN BOLETA)T1
INNER JOIN
(SELECT IDFUNCION, IDOBRA FROM FUNCION NATURAL JOIN OBRA WHERE REALIZADO=1 AND FECHA BETWEEN 'PFecha1' AND 'PFecha2')T2
ON T1.IDFUNCION=T2.IDFUNCION)T3
INNER JOIN
(SELECT IDOBRA FROM COMPANIAOBRA NATURAL JOIN COMPANIA WHERE IDCOMPANIA=PIdCompañia)T4
ON T3.IDOBRA=T4.IDOBRA
ORDER BY POrden;
```

- **Distribución de datos con respecto a parámetros de entrada:**

Compañía: Dado que los datos en boletas tienen clientes aleatorios, y de que existen aproximadamente 1000 clientes y 1000 compañías, no se puede definir una selectividad exacta de cantidad de clientes para una función, o mas aun, para una obra en específico. Así mismo, la conexión entre compañías con obras es aleatoria por lo tanto saber qué compañías tienen que numero de clientes es desconocido.

Rango de fechas: La selectividad de las fechas es inexacta puesto que se agregaron fechas aleatorias entre el rango de 2 años. Aun así, se puede confirmar que las funciones de 2016 son 655 y las de 2017 son 356, por esto, buscar dentro del año 2017 tendrá más selectividad.

- **Casos de prueba:**

Direccion de la consulta: http://localhost:9000/VideoAndes/rest/rfc9

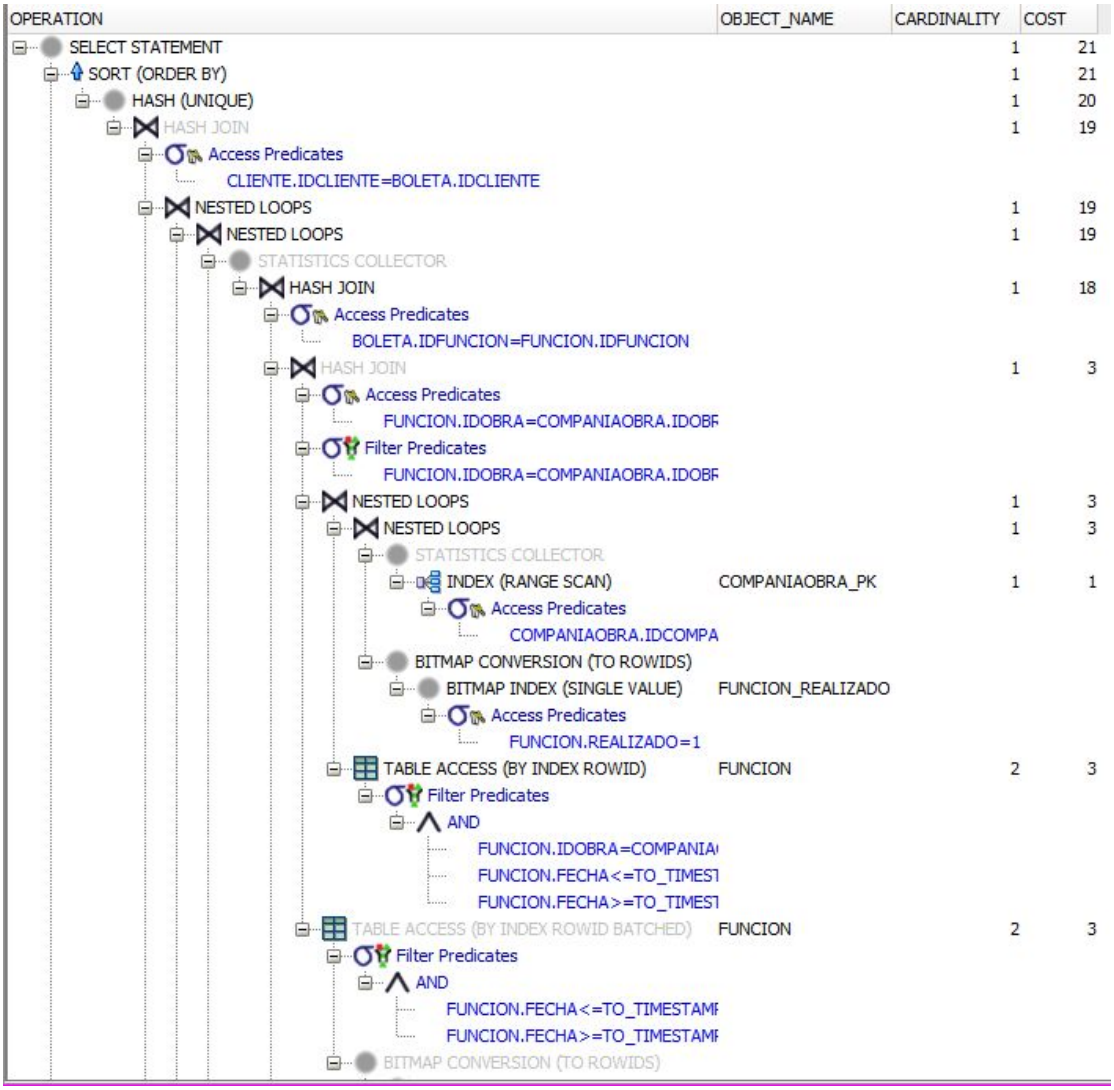
	Analisis 1	Analisis 2
fecha_inicial	31-12-2016	31-12-2016
fecha_final	01-01-2017	31-12-2017
id_compañia	1	996
order_by	IDCLIENTE	IDENTIFICACION

- **Planes de ejecución:**
    - **Resultado de la consulta**

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		4	24
SORT (UNIQUE)		4	23
NESTED LOOPS (SEMI)		71	22
HASH JOIN		179	22
Access Predicates			
BOLETA.IDFUNCION=FUNCION.IDFUNCION			
HASH JOIN		9	20
Access Predicates			
CLIENTE.IDCLIENTE=BOLETA.IDCLIENTE			
TABLE ACCESS (FULL)	BOLETA	9	15
TABLE ACCESS (FULL)	CLIENTE	1003	5
TABLE ACCESS (BY INDEX ROWID BATCHED)	FUNCION	219	2
Filter Predicates			
AND			
FUNCION.FECHA<=TO_TIMESTAMP('31/12/2016')			
FUNCION.FECHA>=TO_TIMESTAMP('01/01/2016')			
BITMAP CONVERSION (TO ROWIDS)			
BITMAP INDEX (SINGLE VALUE)	FUNCION_REALIZADO		
Access Predicates			
FUNCION.REALIZADO=1			
INDEX (UNIQUE SCAN)	COMPANIAOBRA_PK	2	0
Access Predicates			
AND			
COMPANIAOBRA.IDCOMPANIA=1			
FUNCION.IDOBRA=COMPANIAOBRA.IDOBRA			

	IDCLIENTE	NOMBRE	APELLIDO	EMAIL	IDENTIFICACION
1	197	Ellissa	Shervil	eshervil5d@upenn.edu	315-84-3024
2	567	Clem	Pattenden	cpattendenfn@youtube.com	195-92-2380
3	676	Jorge	Dymond	jdymondio@smh.com.au	612-40-1369
4	820	Arlee	Tunney	atunneymo@cmu.edu	743-28-5315

Tiempo consulta 0.20 s
Tiempo plan 0.117s


OPERATION
OBJECT_NAME
CARDINALITY
COST
SELECT STATEMENT 1 21
SORT (ORDER BY) 1 21
HASH (UNIQUE) 1 20
HASH JOIN 1 19
Access Predicates
CLIENTE.IDCLIENTE=BOLETA.IDCLIENTE
NESTED LOOPS 1 19
NESTED LOOPS 1 19
STATISTICS COLLECTOR
HASH JOIN 1 18
Access Predicates
BOLETA.IDFUNCION=FUNCION.IDFUNCION
HASH JOIN 1 3
Access Predicates
FUNCION.IDOBRA=COMPANIAOBRA.IDOBR
Filter Predicates
FUNCION.IDOBRA=COMPANIAOBRA.IDOBR
NESTED LOOPS 1 3
NESTED LOOPS 1 3
STATISTICS COLLECTOR
INDEX (RANGE SCAN) COMPANIAOBRA_PK 1 1
Access Predicates
COMPANIAOBRA.IDCOMPA
BITMAP CONVERSION (TO ROWIDS)
BITMAP INDEX (SINGLE VALUE) FUNCION_REALIZADO
Access Predicates
FUNCION.REALIZADO=1
TABLE ACCESS (BY INDEX ROWID) FUNCION 2 3
Filter Predicates
AND
FUNCION.IDOBRA=COMPANIA
FUNCION.FECHA<=TO_TIMEST
FUNCION.FECHA>=TO_TIMEST
TABLE ACCESS (BY INDEX ROWID BATCHED) FUNCION 2 3
Filter Predicates
AND
FUNCION.FECHA<=TO_TIMESTAM
FUNCION.FECHA>=TO_TIMESTAM
BITMAP CONVERSION (TO ROWIDS)

	IDCLIENTE	NOMBRE	APELLIDO	EMAIL	IDENTIFICACION
1	612	Shaylynn	Haselwood	shaselwoodgw@multiply.com	121-28-4835
2	372	Aksel	McLewd	amclewda8@cbslocal.com	132-37-5930
3	854	Shane	Chaundy	schaundynm@hc360.com	153-86-3842
4	968	Maybelle	Berkeley	mberkeleyqs@sitemeter.com	186-85-4683
5	495	Nehemiah	Yeell	nyeelldn@mlb.com	196-93-8641
6	325	Markos	Colville	mcolville8x@freewebs.com	239-44-7364
7	79	Giffie	Tait	gtait23@goodreads.com	299-69-4813
8	54	Cassy	Orht	corht1e@simplemachines.org	309-81-7182
9	555	Alaster	Drinkale	adrinkalefb@tuttocitta.it	320-56-6663
10	269	Todd	Clutton	tclutton7d@nymag.com	342-00-1237
11	979	Efren	Lavrinov	elavrinovr3@google.ca	347-70-7342
12	239	Simonette	Painten	spainten6j@dailymail.co.uk	360-97-5841
13	246	Shaylynn	Gregh	sgregh6q@blinklist.com	366-50-9655
14	193	Glenda	Jeyness	gjeyness59@mtv.com	377-54-9360
15	484	Mathilda	Matveichev	mmatveichevdc@japanpost.jp	388-63-8335
16	358	Yasmeen	Mathan	ymathan9u@discuz.net	432-46-5576
17	87	Virgina	Quoit	vquoit2b@blog.com	539-63-1433
18	727	Abbi	Stronach	astronachk3@alibaba.com	586-03-6832
19	137	Alden	Kissick	akissick3p@soundcloud.com	589-75-4009
20	799	Cissy	Longfoot	clongfootm3@shareasale.com	592-98-9322
21	229	Rickert	Gelderd	rgelderd69@foxnews.com	613-28-5869
22	342	Ruthy	McAuslan	rmcauslan9e@hatena.ne.jp	642-86-3269

Tiempo plan : 0.027s
Tiempo consulta: 0.013s

- **Plan propuesto:**
- **Plan realizado por ORACLE:** Oracle realiza los JOIN de las tablas por medio de los NESTED LOOPS, que escanea la menor tabla y la guarda en memoria, para iterar luego sobre la tabla más grande, conservando así siempre una tabla en memoria. Para buscar sobre las tablas las tuplas que contienen las condiciones, usa los siguientes algoritmos: SINGLE VALUE: lo utiliza para revisar las funciones que fueron realizadas, ya que todas tienen un valor igual en la columna de realizado. INDEX ROWID: lo utiliza para comparar fechas, ya que la tabla no está particionada y busca las tuplas por el índice primario. RANGE SCAN: lo utiliza para comparar el id de compañia. El algoritmo busca los ids por rango.

RFC10:

- ## Documentación de escenarios de prueba
  - **Sentencia SQL:** Para este requerimiento se modificó la consulta anterior que une las tablas cliente, boleta, función, obra, companiaobra y compañía; que trae los datos de los clientes dada una compañía, que hayan asistido a una función de dicha compañía entre un lapso de tiempo establecido, Finalmente se realiza una selección completa de los clientes que existen y se resta de ella los clientes que aparecieron en la segunda consulta. La consulta se puede ordenar según alguno de los criterios que aparecen en el cliente.

```
SELECT IDCLIENTE, NOMBRE, APELLIDO, EMAIL, IDENTIFICACION FROM CLIENTE
MINUS
SELECT DISTINCT IDCLIENTE, T3.NOMBRE,APELLIDO, T3.EMAIL, IDENTIFICACION FROM
(SELECT IDCLIENTE, T1.NOMBRE, APELLIDO, EMAIL, IDENTIFICACION, IDOBRA FROM
(SELECT IDCLIENTE, NOMBRE, APELLIDO, EMAIL, IDENTIFICACION, IDFUNCION FROM CLIENTE NATURAL JOIN BOLETA)T1
INNER JOIN
(SELECT IDFUNCION, IDOBRA FROM FUNCION NATURAL JOIN OBRA WHERE REALIZADO=1 AND FECHA BETWEEN 'PFecha1' AND 'PFecha2')T2
ON T1.IDFUNCION=T2.IDFUNCION)T3
INNER JOIN
(SELECT IDOBRA FROM COMPANIAOBRA NATURAL JOIN COMPANIA WHERE IDCOMPANIA=PIdCompañia)T4
ON T3.IDOBRA=T4.IDOBRA
ORDER BY POrden;
```

- **Distribución de datos con respecto a parámetros de entrada:**

La selectividad en esta consulta es la misma que en la anterior, dado que ambas consultas realizar una selección con el mismo algoritmo y parámetros, solo que esta toma esas tuplas y se las quita a la tabla general de clientes

- **Casos de prueba:**

Direccion de la consulta: http://localhost:9000/VideoAndes/rest/rfc9

	Analisis 1	Analisis 2
fecha_inicial	31-12-2016	31-12-2016
fecha_final	01-01-2017	31-12-2017
id_compañia	1	996
order_by	IDCLIENTE	IDENTIFICACION

- **Planes de ejecución:**
  - **Resultado de la consulta**

	IDCLIENTE	NOMBRE	APELLIDO	EMAIL	IDENTIFICACION
1	189	Stoddard	Marquot	smarquot55@tumblr.com	101-16-3334
2	19	Lee	Rymour	lrymourf@paypal.com	101-39-6092
3	1	ZULMA	CASTAÑEDA	ZL.CASTANEDA@GMAIL.COM	1015450052
4	2	CAMILO	USECHE	JC.USECHE@GMAIL.COM	1015450052
5	164	Pip	Schuricht	pschuricht4g@soup.io	103-72-9719
6	520	Frieda	Gunbie	fgunbieec@usnews.com	103-76-5402
7	922	Dexter	Cuvley	dcuvleypi@sciencedirect.com	104-67-0554
8	147	Berna	Alphege	balphege3z@hp.com	106-99-1328
9	223	Sidnee	Laurent	slaurent63@economist.com	107-70-9965
10	57	Stephenie	Golder	sgolder1h@newyorker.com	111-02-1153
11	785	Edik	Du Hamel	eduhamellp@people.com.cn	113-92-6223
12	904	Farica	Schwanden	fschwandenp0@delicious.com	114-53-9707

Tiempo consulta: 0.17s

Tiempo plan: 0.129s

	IDCLIENTE	NOMBRE	APELLIDO	EMAIL	IDENTIFICACION
1	1	ZULMA	CASTAÑEDA	ZL.CASTANEDA@GMAIL.COM	1015450052
2	2	CAMILO	USECHE	JC.USECHE@GMAIL.COM	1015450052
3	3	JUAN	LUDWING	LUDWINGTHEHUNTER@GMAIL.COM	800145621
4	4	Elyssa	Ding	eding0@cpanel.net	885-06-6458
5	5	Chiquia	Burghall	cburghall1@nhs.uk	868-27-9270
6	6	Kanya	Cowgill	kcowgill2@cafepress.com	472-63-8414
7	7	Isidore	Crowest	icrowest3@chron.com	412-70-3215
8	8	Glenn	Goulding	ggoulding4@surveymonkey.com	332-80-2400
9	9	Dannye	Taleworth	dtaleworth5@dagondesign.com	182-70-8707
10	10	Lambert	Mewis	lmewis6@trellian.com	743-37-6922
11	11	Dyanna	Pigott	dpigott7@shutterfly.com	915-12-6764
12	12	Norton	Amorine	namorine8@friendfeed.com	493-07-5695
13	13	Ugo	Rawlison	urawlison9@dedecms.com	619-00-6456
14	14	Wiley	Kaasman	wkaasmana@redcross.org	222-22-6484
15	15	Darla	Pledger	dpledgerb@prnewswire.com	373-14-3380

Tiempo consulta: 0.157
Tiempo plan: 0.168

- **Plan propuesto:**

Se espera el uso de HASH JOIN para unir las tablas involucradas y el uso del índice creado FUNCION_REALIZADO para encontrar rápidamente las funciones que sí fueron realizadas.
Se espera además que haya un acceso completo a la tabla de clientes para poder realizar la resta con la consulta anterior.

- **Plan realizado por ORACLE:**

ORACLE realiza como era esperado una consulta usando HASH JOIN y NESTED LOOPS a diferencia del caso anterior se debía realizar un ordenamiento más para poder hacer la resta con el resultado anterior. Se observa el uso del índice BITMAP creado (FUNCION_REALIZADO) para encontrar con mayor rapidez las funciones realizadas. Cabe tambien resaltar que ORACLE realiza un un UNIQUE SCAN sobre el id (llave primaria) de la tabla compañia, pues el dato que se busca es exacto y esta sobre el índice COMPANIAOBRA_PK. Nos llamo la atencion que este mismo indice se utilizara de manera RANGE SCAN para facilitar los JOIN.

RFC11:

- **Documentación de escenarios de prueba**

Sentencia SQL: Para este requerimiento se entregan los valores de nombre de la obra, la fecha de la función, el sitio de la función, el conteo de boletas vendidas para esa función y la cantidad de usuarios que compraron boletas de esa función. Las tuplas cumplen criterios como el día y hora y fecha de la función, la descripción de los elementos del espacio y la localidad de las boletas contadas.

```
SELECT
NOMBREOBRA AS NOMBRE_OBRA,
FECHA AS FECHA_FUNCION,
NOMBRE AS SITIO_FUNCION,
COUNT(IDBOLETA) AS CANTIDAD_BOLETAS_VENDIDAS,
COUNT(IDCLIENTE) AS CANTIDAD_USUARIOS_REGISTRADOS
FROM
OBRA
NATURAL JOIN
FUNCION
NATURAL JOIN
ESPACIO
NATURAL JOIN
BOLETA
NATURAL JOIN
CLIENTE
NATURAL JOIN
LOCALIDAD
NATURAL JOIN
DESCRIPCIONESESPACIO
NATURAL JOIN
DESCRIPCIONTECNICA
WHERE
FECHA BETWEEN 'PFecha1' AND 'PFecha2'
AND
DESCRIPCION LIKE '%PBusqueda%'
AND
NOMBRELOCALIDAD = 'PLocalidad'
AND
```

```
TO_CHAR(FECHA, 'HH24') BETWEEN 'PHora1' AND 'PHora2'
AND
TO_CHAR(FECHA, 'DAY',  'NLS_DATE_LANGUAGE=ENGLISH') LIKE '%PDia%'
GROUP BY NOMBREOBRA, FECHA, NOMBRE;
```

- **Distribución de datos con respecto a parámetros de entrada:**

Fecha_inicial y Fecha_final: La distribución de los datos en las fechas es completamente aleatorio debido a la manera en que se insertaron los datos. Sin embargo hay 655 funciones en el 2016 y 356 funciones en el 2017, lo cual incrementa el porcentaje de funciones realizadas (lo cual nos interesa para realizar nuestras pruebas.

Descripción: Se utilizó texto dummy Lorem ipsum para llenar las descripciones de las obras, este texto no cuenta con palabras repetidas muy a menudo por lo tanto la selectividad de la consulta resulta siendo alta. Pero no puedo decirla con exactitud.

Nombre_localidad: A pesar de que solo existen 4 localidades, todas ellas difieren según el sitio donde se presenta la obra. Sin embargo, garantizamos que la distribución de tipos de localidad fuera completamente uniforme (251 para cada uno). Esto garantiza la efectividad del índice BITMAP planteado.

H**ora_inicio y Hora_fin:** Al estar relacionado directamente con la fecha de la función, la distribución es similar al caso de arriba (aunque debería ser cercana al 50/50) pues hay la misma cantidad de horas en la mañana que en la tarde.

Dia: Al igual que el caso anterior, se encuentran distribuidas uniformemente.

Cabe resaltar que al combinar todas las condiciones, conseguimos una selectividad elevada pues a pesar del volumen de datos, encontrar tuplas que cumplan con todas las condiciones es complicado.

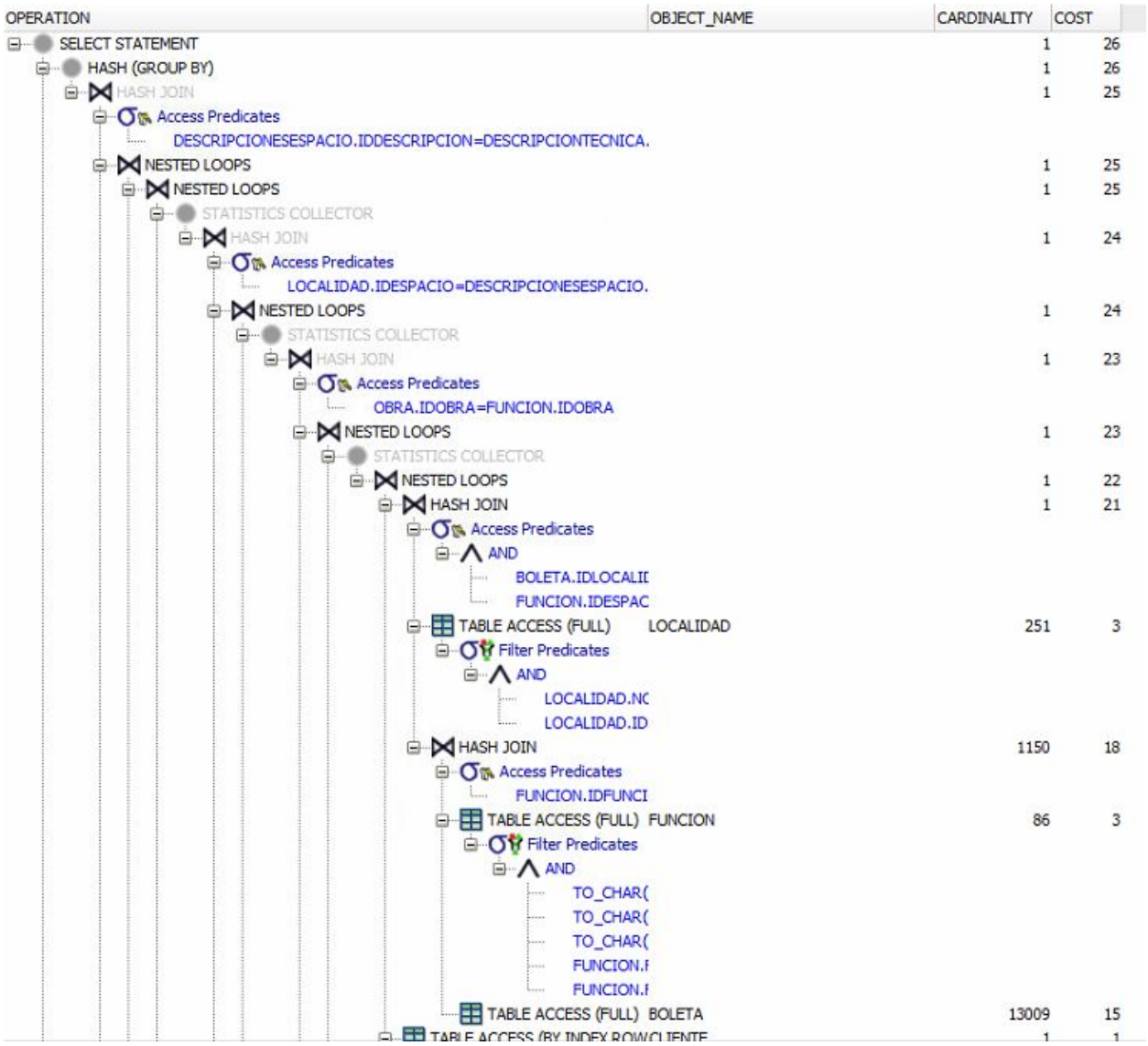
- **Casos de prueba:**

Direccion de la consulta: http://localhost:9000/VideoAndes/rest/rfc9

	Analisis 1	Analisis 2
fecha_inicial	01-01-2016	01-01-2017
fecha_final	31-12-2016	31-12-2017
descripción	Lorem	eget
nombre_localidad	VIP	VIP
hora_inicio	01	12
hora_fin	13	23
dia	MONDAY	SATURDAY

- **Planes de ejecución:**

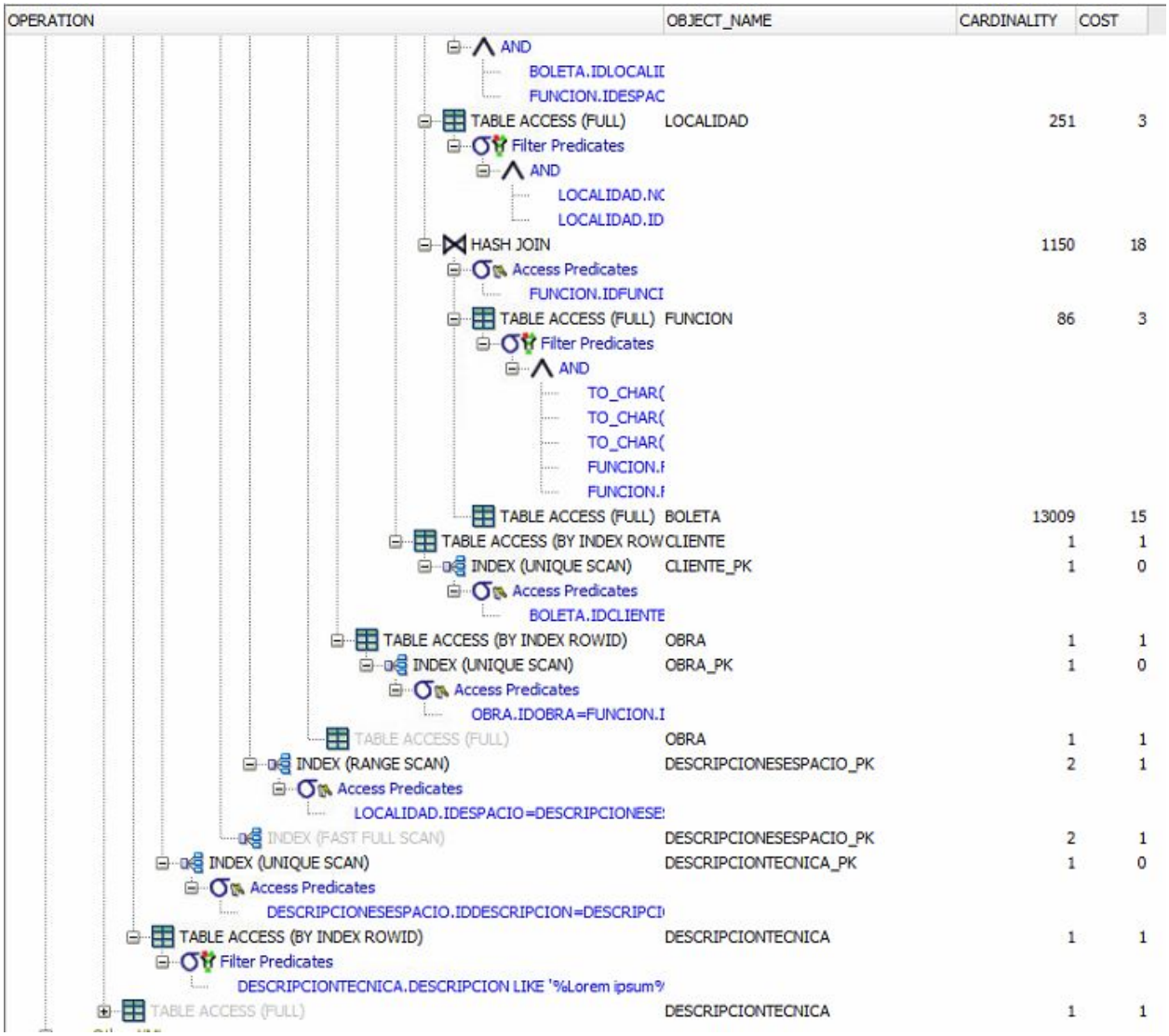
	NOMBRE_OBRA	FECHA_FUNCION	SITIO_FUNCION	CANTIDAD_BOLETAS_VENDIDAS	CANTIDAD_USUARIOS_REGISTRADOS
1	risus dapibus augue	26/12/16 12:14:22,000000000 AM	Sallee	1	1
2	risus dapibus augue	26/12/16 12:14:22,000000000 AM	Kimberlyn	1	1
3	risus dapibus augue	26/12/16 12:14:22,000000000 AM	Sybila	1	1
4	risus dapibus augue	26/12/16 12:14:22,000000000 AM	Maire	1	1
5	risus dapibus augue	26/12/16 12:14:22,000000000 AM	Eolande	1	1
6	risus dapibus augue	26/12/16 12:14:22,000000000 AM	Morley	1	1
7	risus dapibus augue	26/12/16 12:14:22,000000000 AM	Lebbie	1	1
8	risus dapibus augue	26/12/16 12:14:22,000000000 AM	Gina	1	1
9	risus dapibus augue	26/12/16 12:14:22,000000000 AM	Shaylynn	1	1
10	risus dapibus augue	26/12/16 12:14:22,000000000 AM	Nola	1	1
11	risus dapibus augue	26/12/16 12:14:22,000000000 AM	Humbert	1	1
12	risus dapibus augue	26/12/16 12:14:22,000000000 AM	Amargo	1	1
13	risus dapibus augue	26/12/16 12:14:22,000000000 AM	Gordy	1	1
14	risus dapibus augue	26/12/16 12:14:22,000000000 AM	Zulema	1	1



Tiempo consulta: 0.095s

Tiempo plan: 0.101 s

	NOMBRE_OBRA	FECHA_FUNCION	SITIO_FUNCION	CANTIDAD_BOLETAS_VENDIDAS	CANTIDAD_USUARIOS_REGISTRADOS
1	velit id	14/01/17 04:12:49,000000000 PM	Darcy	1	1
2	velit id	14/01/17 04:12:49,000000000 PM	Lebbie	1	1
3	velit id	14/01/17 04:12:49,000000000 PM	Lee	1	1
4	velit id	14/01/17 04:12:49,000000000 PM	Gris	1	1
5	in hac habitasse platea dictumst	11/03/17 06:02:50,000000000 PM	Lay	1	1
6	in hac habitasse platea dictumst	11/03/17 06:02:50,000000000 PM	Vere	2	2
7	in hac habitasse platea dictumst	11/03/17 06:02:50,000000000 PM	Sherman	1	1
8	in hac habitasse platea dictumst	11/03/17 06:02:50,000000000 PM	Arielle	1	1
9	in hac habitasse platea dictumst	11/03/17 06:02:50,000000000 PM	Leeann	1	1
10	in hac habitasse platea dictumst	11/03/17 06:02:50,000000000 PM	Rosalie	1	1
11	in hac habitasse platea dictumst	11/03/17 06:02:50,000000000 PM	Sibyl	1	1
12	velit id	14/01/17 04:12:49,000000000 PM	Wendeline	1	1



Tiempo consulta: 0.156s

Tiempo plan: 0.101s

- **Plan propuesto:**

Como en este requerimiento se realizaron muchos natural join se esperaba que oracle realizará un HASH JOIN para reducir el costo de las uniones, también se espera el uso de los indices sobre las llaves primarias para localizar rápidamente las condiciones impuestas en el where por cada tabla (ejemplo la búsqueda del rango de fechas/horas)

- **Plan realizado por ORACLE:**

ORACLE no utilizó el índice propuesto por nosotros en la columna de fecha debido a la expresión LIKE que aparece en el WHERE, creíamos que esto sería útil en campos como la búsqueda del dia de la semana (‘MONDAY’, ‘TUESDAY’ etc) pero por alguna razón no podían ser encontrados con la expresión ‘=’ así que nos vimos obligados a utilizar la expresión LIKE que no se beneficia por el uso de índices.

Para los JOINS, ORACLE hizo uso de los esperados HASH JOIN pero además de los NESTED LOOPS, al parecer debido a la gran cantidad de datos que deben unirse, ORACLE prefiere realizar los JOINS de las tablas una por una para tener siempre una tabla grande que ir escaneando y una tabla pequeña que si le cabe en memoria. Así que la principal diferencia entre lo esperado y lo obtenido fue el uso de NESTED LOOPS.

RFC12:

- **Documentación de escenarios de prueba**

- **Sentencia SQL:** En esta consulta se solicita retornar los mejores clientes, es decir aquellos que han comprado más de N cantidad de boletas en la localidad VIP dando la posibilidad al usuario de elegir cuanto será ese N

```
SELECT IDCLIENTE,
NOMBRE,
APELLIDO,
EMAIL,
IDENTIFICACION,
CONTEO
FROM
(SELECT
IDCLIENTE,
NOMBRE,
APELLIDO,
EMAIL,
IDENTIFICACION,
COUNT(IDBOLETA) AS CONTEO
FROM
(CLIENTE
NATURAL JOIN
BOLETA
NATURAL JOIN
LOCALIDAD)
WHERE NOMBRELOCALIDAD = 'VIP'
GROUP BY IDCLIENTE, NOMBRE, APELLIDO, EMAIL, IDENTIFICACION)
WHERE CONTEO >= PNumero;
```

- **Distribución de datos con respecto a parámetros de entrada:**

El único dato de entrada para esta consulta es Conteo (N) el cual se utiliza para saber cuántas personas han comprado más de esa cantidad de boletas en VIP. Este parámetro hace variar la selectividad de las consultas dependiendo de la cantidad que se pida, pues pocas personas han comprado muchas boletas en VIP, pero una cantidad más considerable ha comprado al menos 1.

- **Casos de prueba:**

Direccion de la consulta: http://localhost:9000/VideoAndes/rest/rfc9

	Analisis 1	Analisis 2
conteo	5	7

- **Planes de ejecución:**

	IDCLIENTE	NOMBRE	APELLIDO	EMAIL	IDENTIFICACION	CONTEO
1	550	Candide	Marlin	cmarlinf6@ibm.com	313-48-0882	16
2	282	Ken	Brittle	kbrittle7q@rambler.ru	181-70-7014	13
3	941	Oliver	Courcey	ocourceyq1@state.tx.us	883-19-8100	14
4	531	Burke	Underhill	bunderhillen@toplist.cz	229-99-2810	12
5	221	Odella	Ladloe	oladloe61@xing.com	912-55-2524	8
6	794	Willard	Rigden	wrigdenly@sfgate.com	942-53-3649	11
7	789	Brana	Seczyk	bseczyklt@deviantart.com	519-01-1364	13
8	853	Gaby	Cordon	gcordonnl@gravatar.com	965-31-4012	12
9	256	Kenna	Bront	kbront70@vk.com	698-35-7693	10
10	33	Marty	Paintain	mpaintaint@surveymonkey.com	277-77-3259	15
11	647	Kristofor	Thackeray	kthackerayhv@paginegialle.it	231-02-7953	14
12	59	Gwynne	Huriche	ghuriche1j@bloglines.com	233-48-4889	7
13	333	Emmeline	Slingsby	eslingsby95@wiley.com	248-74-3328	11
14	194	Saxe	Godrich	sgodrich5a@gravatar.com	179-40-5570	22
15	746	Darline	Scargill	dscargillkm@live.com	527-15-1916	7
16	714	Lea	Bech	lbechjq@lycos.com	783-71-3437	6
17	618	Geraldine	Brockman	gbrockmanh2@baidu.com	958-87-4964	19
18	87	Virgina	Quoit	vquoit2b@blog.com	539-63-1433	13
19	188	Ronna	Snuggs	rsnuggs54@cpanel.net	785-49-8051	14

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		50	24
FILTER			
Filter Predicates			
COUNT(*)>=5			
HASH (GROUP BY)		50	24
HASH JOIN		4476	23
Access Predicates			
CLIENTE.IDCLIENTE=BOLETA.IDCLIENTE			
TABLE ACCESS (FULL)	CLIENTE	1003	5
HASH JOIN		4476	18
Access Predicates			
BOLETA.IDLOCALIDAD=LOCALIDAD.IDLOCALIDAD			
TABLE ACCESS (FULL)	LOCALIDAD	251	3
Filter Predicates			
LOCALIDAD.NOMBRELOCALIDAD='VIP'			
TABLE ACCESS (FULL)	BOLETA	13009	15

Tiempo consulta: 0.049s
Tiempo plan: 0.072s

	IDCLIENTE	NOMBRE	APELLIDO	EMAIL	IDENTIFICACION	CONTEO
1	550	Candide	Marlin	cmarlinf6@ibm.com	313-48-0882	16
2	282	Ken	Brittle	kbrittle7q@rambler.ru	181-70-7014	13
3	941	Oliver	Courcey	ocourceyq1@state.tx.us	883-19-8100	14
4	531	Burke	Underhill	bunderhillen@toplist.cz	229-99-2810	12
5	221	Odella	Ladloe	oladloe61@xing.com	912-55-2524	8
6	794	Willard	Rigden	wrigdenly@sfgate.com	942-53-3649	11
7	789	Brana	Seczyk	bseczyklt@deviantart.com	519-01-1364	13
8	853	Gaby	Cordon	gcordonnl@gravatar.com	965-31-4012	12
9	256	Kenna	Bront	kbront70@vk.com	698-35-7693	10
10	33	Marty	Paintain	mpaintaint@surveymonkey.com	277-77-3259	15
11	647	Kristofor	Thackeray	kthackerayhv@paginegialle.it	231-02-7953	14
12	59	Gwynne	Huriche	ghuriche1j@bloglines.com	233-48-4889	7
13	333	Emmeline	Slingsby	eslingsby95@wiley.com	248-74-3328	11
14	194	Saxe	Godrich	sgodrich5a@gravatar.com	179-40-5570	22
15	746	Darline	Scargill	dscargillkm@live.com	527-15-1916	7
16	618	Geraldine	Brockman	gbrockmanh2@baidu.com	958-87-4964	19

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		50	24
FILTER			
Filter Predicates			
COUNT(*)>=7			
HASH (GROUP BY)		50	24
HASH JOIN		4476	23
Access Predicates			
CLIENTE.IDCLIENTE=BOLETA.IDCLIENTE			
TABLE ACCESS (FULL)	CLIENTE	1003	5
HASH JOIN		4476	18
Access Predicates			
BOLETA.IDLOCALIDAD=LOCALIDAD.IDLOCALIDAD			
TABLE ACCESS (FULL)	LOCALIDAD	251	3
Filter Predicates			
LOCALIDAD.NOMBRELOCALIDAD='VIP'			
TABLE ACCESS (FULL)	BOLETA	13009	15

Tiempo consulta: 0.098s
Tiempo plan: 0.063s

- **Plan propuesto:**
  Se espera que la base de datos realice un HASH JOIN de las tablas involucradas para luego realizar una búsqueda completa debido a la condición del WHERE que no es específica sino de rango.
- **Plan realizado por ORACLE:** Oracle realiza un hash join entre las tablas de: Cliente, realizando un acceso completo; Localidad, realizando un acceso completo y Boleta, realizando un acceso completo y filtrando por las boletas que son VIP. Al final hace un filtro sobre la tabla resultante, dejando como resultado las tuplas que tengan un conteo de 5/7 o más boletas. Este conteo completo es necesario pues la condición del WHERE no es un = sino un >=

● Diseño del escenario de pruebas de eficiencia

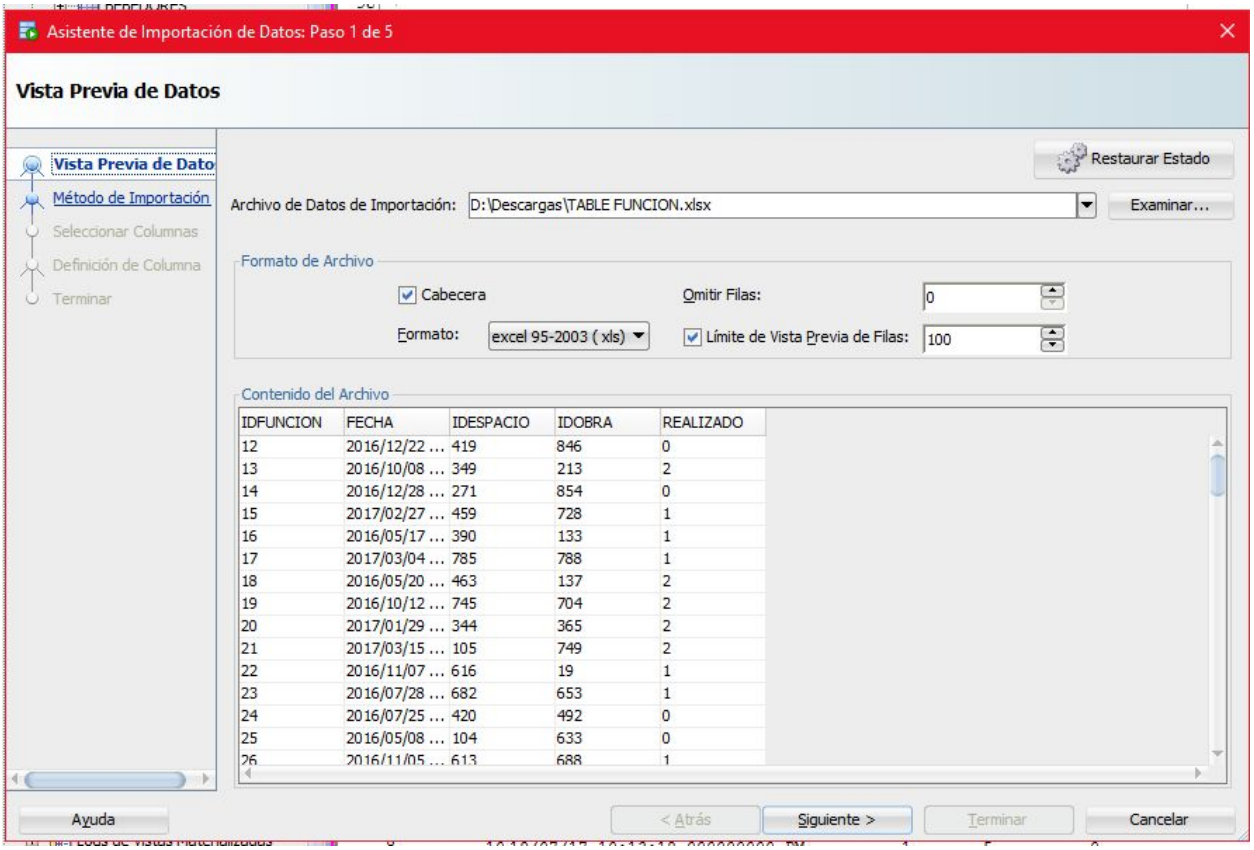
Para llenar las tablas CLIENTE, ESPACIO, LOCALIDAD, OBRA, FUNCION, BOLETA Y COMPANIA, se generaron datos aleatorios utilizando la página web mockaroo.com. Valiéndonos de las funciones para datos mock que la página ofrece logramos crear información que resultara coherente para estas tablas manteniendo y garantizando una distribución uniforme para cada una de ellas. Para complementar esta herramienta y añadir mas personalizacion a los datos insertados utilizamos funciones en Excel.

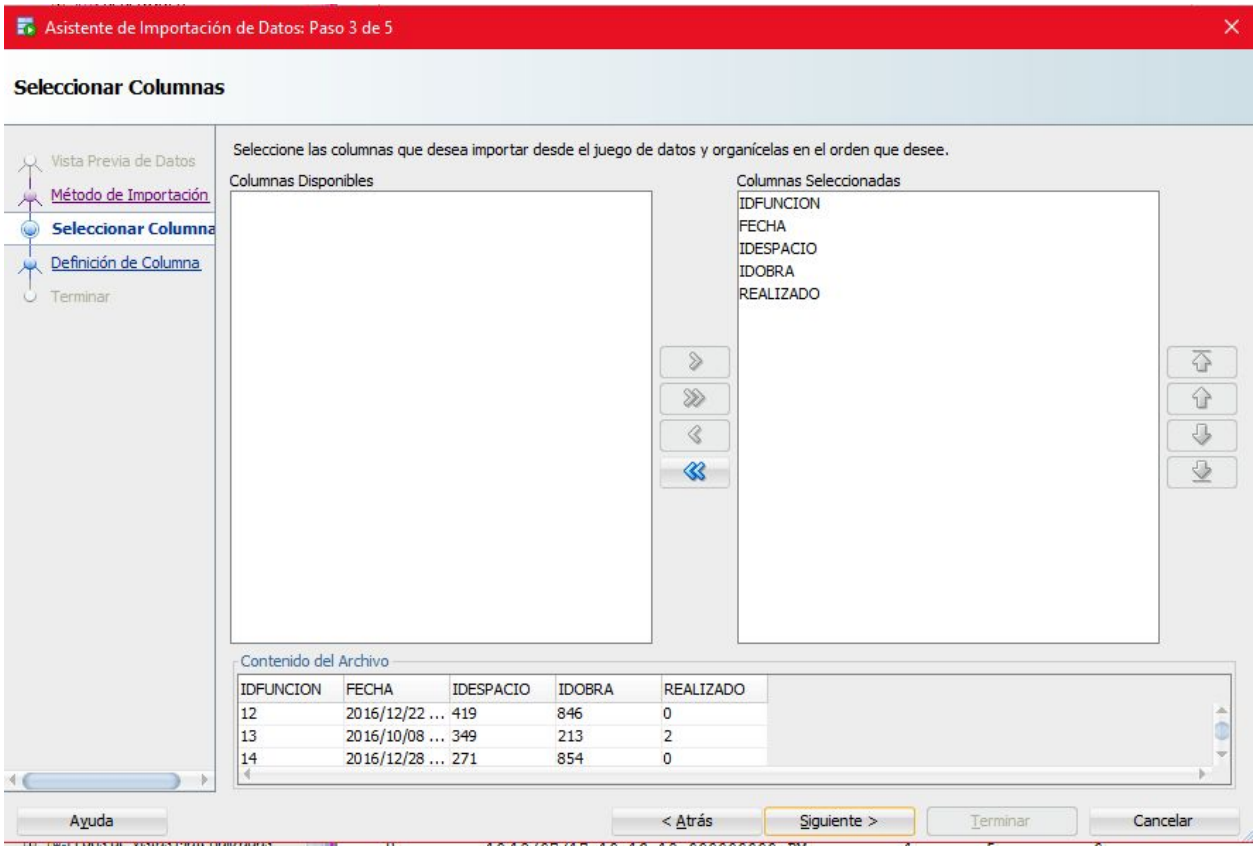
Se procuró mantener siempre la lógica entre los datos, que la fecha de inicio fuera menor a la fecha de fin de una obra, que las boletas fueran compradas antes de la fecha de inicio de la respectiva función, que cada cliente tuviera una variedad de boletas y localidades etc.

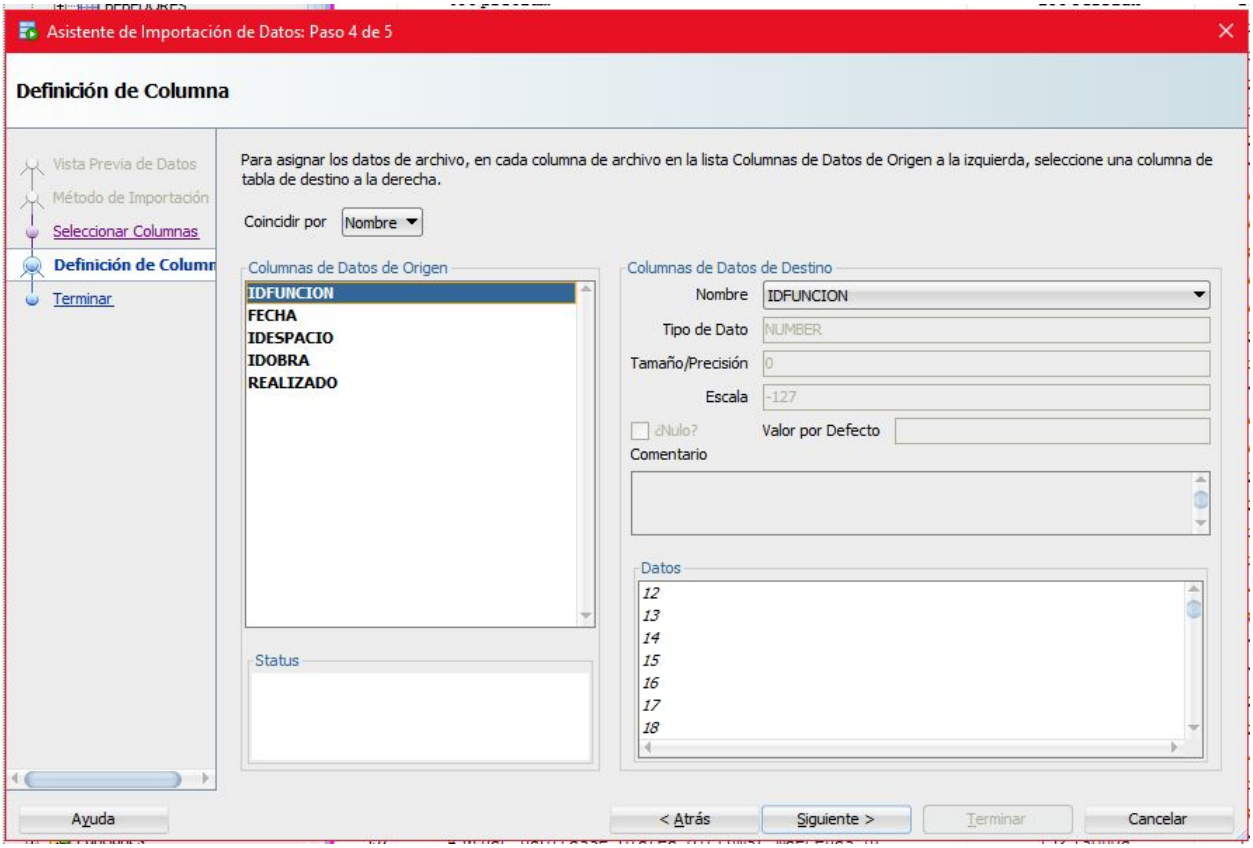
La información que no era tan relevante para el buen resultado de las consultas se generó de manera aleatoria, por ejemplo para el nombre y descripción de las obras de teatro se utilizó *Lorem Ipsum*.

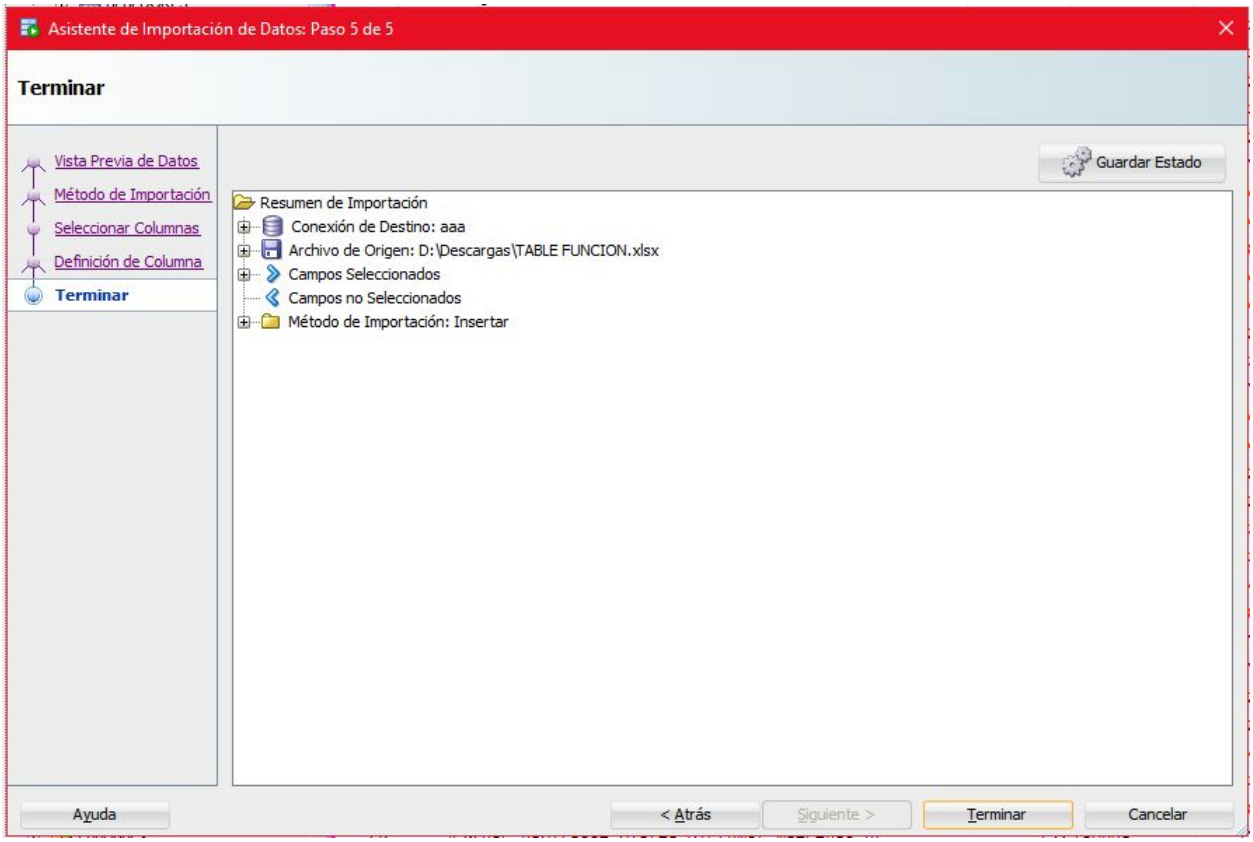
Estos datos, ordenados y generados en archivos *.xlsx* pueden ser importados por medio de SQL Developer, que reconoce la información del archivo y la asigna columna a columna según el usuario desee, como se ve a continuación:

LOS DATOS DE ESTAS TABLAS SE ENCUENTRAN EN LA CARPETA /docs DE LA APLICACIÓN


Asistente de Importación de Datos: Paso 1 de 5
Vista Previa de Datos
Vista Previa de Dato
Método de Importación
Seleccionar Columnas
Definición de Columna
Terminar
Restaurar Estado
Archivo de Datos de Importación: D:\Descargas\TABLE FUNCION.xlsx
Examinar...
Formato de Archivo
Cabecera
Omitir Filas: 0
Formato: excel 95-2003 (xls)
Límite de Vista Previa de Filas: 100
Contenido del Archivo
IDFUNCION FECHA IDESPACIO IDOBRA REALIZADO
12 2016/12/22 ... 419 846 0
13 2016/10/08 ... 349 213 2
14 2016/12/28 ... 271 854 0
15 2017/02/27 ... 459 728 1
16 2016/05/17 ... 390 133 1
17 2017/03/04 ... 785 788 1
18 2016/05/20 ... 463 137 2
19 2016/10/12 ... 745 704 2
20 2017/01/29 ... 344 365 2
21 2017/03/15 ... 105 749 2
22 2016/11/07 ... 616 19 1
23 2016/07/28 ... 682 653 1
24 2016/07/25 ... 420 492 0
25 2016/05/08 ... 104 633 0
26 2016/11/05 ... 613 688 1
Ayuda
< Atrás
Siguiente >
Terminar
Cancelar


Asistente de Importación de Datos: Paso 3 de 5
Seleccionar Columnas
Vista Previa de Datos
Método de Importación
Seleccionar Columna
Definición de Columna
Terminar
Seleccione las columnas que desea importar desde el juego de datos y organícelas en el orden que desee.
Columnas Disponibles
Columnas Seleccionadas
IDFUNCION
FECHA
IDESPACIO
IDOBRA
REALIZADO
Contenido del Archivo
IDFUNCION FECHA IDESPACIO IDOBRA REALIZADO
12 2016/12/22 ... 419 846 0
13 2016/10/08 ... 349 213 2
14 2016/12/28 ... 271 854 0
Ayuda
< Atrás
Siguiente >
Terminar
Cancelar


Asistente de Importación de Datos: Paso 4 de 5
Definición de Columna
Vista Previa de Datos
Método de Importación
Seleccionar Columnas
Definición de Column
Terminar
Para asignar los datos de archivo, en cada columna de archivo en la lista Columnas de Datos de Origen a la izquierda, seleccione una columna de tabla de destino a la derecha.
Coincidir por
Nombre
Columnas de Datos de Origen
IDFUNCION
FECHA
IDESPACIO
IDOBRA
REALIZADO
Columnas de Datos de Destino
Nombre
IDFUNCION
Tipo de Dato
NUMBER
Tamaño/Precisión
0
Escala
-127
¿Nulo?
Valor por Defecto
Comentario
Datos
12
13
14
15
16
17
18
Status
Ayuda
< Atrás
Siguiente >
Terminar
Cancelar


Asistente de Importación de Datos: Paso 5 de 5
Terminar
Vista Previa de Datos
Método de Importación
Seleccionar Columnas
Definición de Columna
Terminar
Guardar Estado
Resumen de Importación
Conexión de Destino: aaa
Archivo de Origen: D:\Descargas\TABLE FUNCION.xlsx
Campos Seleccionados
Campos no Seleccionados
Método de Importación: Insertar
Ayuda
< Atrás
Siguiente >
Terminar
Cancelar

- Análisis del proceso de optimización y el modelo de ejecución de consultas

Analice la diferencia entre la ejecución de consultas delegada al manejador de bases de datos como Oracle y compárelo con una ejecución donde la aplicación trae los datos a memoria principal y resuelve con instrucciones de control (if, while, etc.), los operadores involucrados en las consultas como joins, selecciones y proyecciones.

Los sistemas gestores de bases de datos SGBD como ORACLE realiza su ejecución de consultas a un nivel más bajo (nativo para el sistema de datos), por esta razón ORACLE es mucho más eficiente a la hora de procesar grandes volúmenes de información que lo que podría serlo una aplicación utilizando estructuras de control como if o while. Entre las funciones implementadas por el sistema gestor que contribuyen a la eficiencia del manejo de datos tenemos por ejemplo los índices, que ayudan a buscar rápidamente la información almacenada en memoria secundaria. Otra de las ventajas que ofrece la base de datos es que puede trabajar eficientemente con información en la memoria secundaria mientras que la aplicación debería necesariamente trabajarla en memoria principal.

Para una aplicación seria demasiado costoso traer toda la información almacenada para procesarla, considerando que su volumen sea muy elevado, resulta mucho más efectivo para la aplicación trabajar únicamente sobre los datos necesarios. Además, entre menos accesos directos al lugar donde la información se encuentra almacenada tenga que hacer la aplicación es mejor en términos de tiempo de respuesta para el usuario.

Por último, y teniendo en cuenta que en la vida real en muchas ocasiones se deben procesar datos del orden de millones (y en poco tiempo) realizar este procesamiento en la aplicación sería muy inefectivo pues, y a pesar de que siempre procuramos mantener la complejidad de los algoritmos al mínimo, hay muchas operaciones que no se pueden realizar en menor complejidad (sea cuadrática, cúbica o incluso superior). Sin embargo la base de datos está diseñada específicamente para soportar esta cantidad de información procesando consultas bastante complejas gracias el lenguaje SQL.

En conclusión podemos afirmar que lo mejor es que cada cosa haga el trabajo que mejor sabe hacer, la aplicación se encarga de procesar niveles pequeños de información y brindarle al programador un mayor nivel de control sobre lo que ocurre y la interacción con el usuario, La base de datos se encarga de manejar eficientemente grandes volúmenes de información respondiendo a las preguntas de la aplicación con los datos que esta requiere específicamente.