

Iteración 3

Sebastián García 201630047, Nicolas Sotelo 201623026

Grupo D-05

Sistemas Transaccionales

Ingeniería de Sistemas y Computación

{js.garcia1, n.sotelo}@uniandes.edu.co

Mayo 20 de 2018

Tabla de contenido

1	Introducción	1
2	Diseño y Construcción de la Aplicación	1
2.1	Diseño de la aplicación	1
3	Diseño físico de la Aplicación	2
3.1	Índices generados por Oracle	2
3.2	RFC 10 Consultar consumo en Alohandes	3
3.3	RFC 11 Consultar inversa de consumo en Alohandes	4
3.4	RFC 12 Consultar de funcionamiento	6
3.5	RFC 13 Consultar los buenos clientes	7
4	Diseño de datos	10
5	Análisis del proceso de optimización y el modelo de ejecución de consultas	11
6	Escenarios de prueba	11
6.1	RFC 10 y 11 Consumo Alohandes	11
7	Consideraciones	16
8	Resultados	16

1 Introducción

El objetivo de la siguiente iteración es integrar requerimientos de eficiencia a una aplicación transaccional desarrollada en una arquitectura de tres niveles, con interfaz mediante servicios REST y manejo de persistencia en base de datos.

2 Diseño y Construcción de la Aplicación

2.1 Diseño de la aplicación

Para realizar los requerimientos funcionales de la presente iteración 3 no se hicieron modificaciones a las relaciones que se implementaron en la anterior iteración 2. De la misma forma no se implementaron nuevas relaciones a las ya establecidas en la anterior iteración.

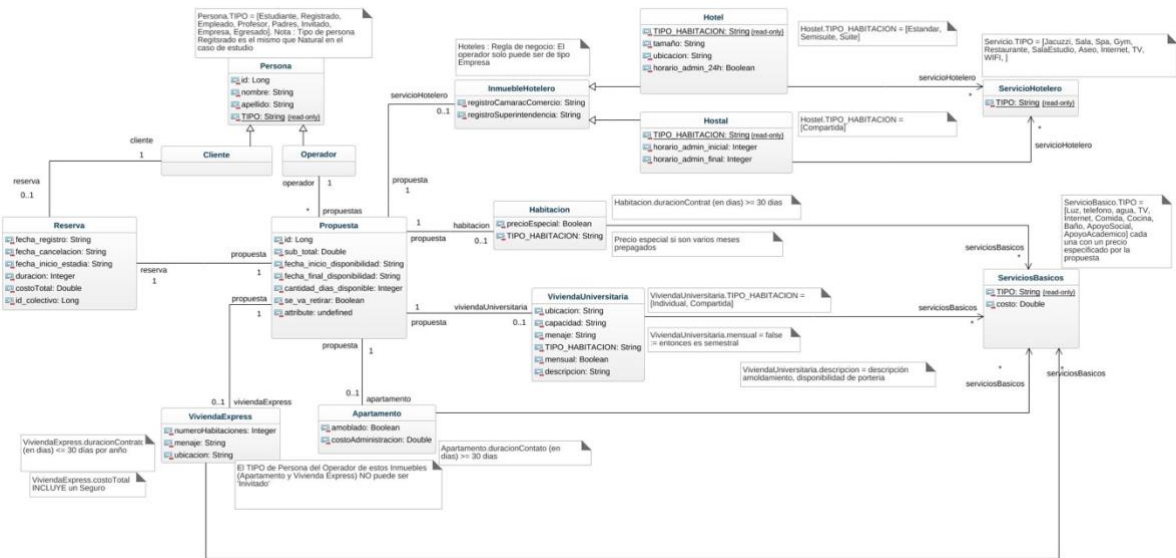


Figura 1. Modelo de clases UML. (Los modelos en formato PDF y original se encuentra en docs > Iteración 3 > modelos).

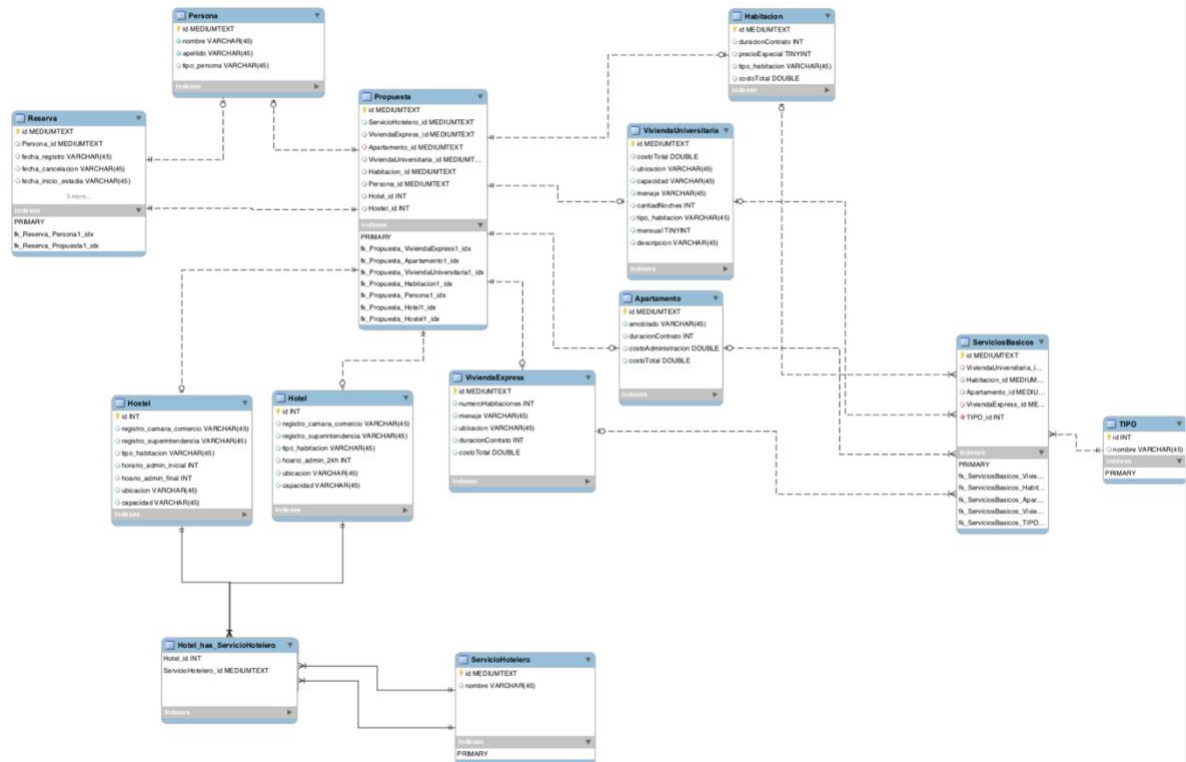


Figura 2. Modelo relacional. (Los modelos en formato PDF y original se encuentra en docs > Iteración 3 > modelos).

3 Diseño físico de la Aplicación

3.1 Índices generados por Oracle

Todos los índices encontrados son creados de forma automática por Oracle debido a que son o hacen parte de una llave primaria esto pasa porque la llave primaria tiene un valor único que me permite identificar cada uno de los registros y encontrarlos en disco de manera más rápida

y eficiente. Estos índices facilitan la realización de *joins* e inserciones en las tablas para los requerimientos anteriores.

SELECT * FROM ALL_IND_COLUMNS WHERE INDEX_OWNER = 'ISIS2304A331810' AND (INDEX_NAME LIKE '%PK' OR INDEX_NAME LIKE 'SYS%');

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	COLUMN_LENGTH	CHAR_LENGTH	DESCEND
1	ISIS2304A331810	SYS_C00322402	ISIS2304A331810	RESERVAS	ID	1	22	0	ASC
2	ISIS2304A331810	SYS_C00322394	ISIS2304A331810	PROPUESTAS	ID	1	22	0	ASC
3	ISIS2304A331810	SYS_C00322388	ISIS2304A331810	SERVICIOS_BASICOS	ID	1	22	0	ASC
4	ISIS2304A331810	SYS_C00322387	ISIS2304A331810	TIPOS	ID	1	22	0	ASC
5	ISIS2304A331810	SYS_C00322386	ISIS2304A331810	HABITACIONES	ID	1	22	0	ASC
6	ISIS2304A331810	SYS_C00322385	ISIS2304A331810	VIVIENDAS_UNIVERSITARIAS	ID	1	22	0	ASC
7	ISIS2304A331810	SYS_C00322383	ISIS2304A331810	VIVIENDAS_EXPRESS	ID	1	22	0	ASC
8	ISIS2304A331810	SYS_C00322379	ISIS2304A331810	SERVICIOS_BASICOS_HOTELEROS	ID	1	22	0	ASC
9	ISIS2304A331810	SYS_C00322372	ISIS2304A331810	HOSTELES	ID	1	22	0	ASC
10	ISIS2304A331810	SYS_C00322367	ISIS2304A331810	HOTELES	ID	1	22	0	ASC
11	ISIS2304A331810	SYS_C00322364	ISIS2304A331810	PERSONAS	ID	1	22	0	ASC
12	ISIS2304A331810	APARTAMENTOS_PK	ISIS2304A331810	APARTAMENTOS	ID	1	22	0	ASC

Figura 3. Índices creados por Oracle.

3.2 RFC 10 Consultar consumo en Alohandes

- Selección de índices

Para este requerimiento se implementaron dos índices. El primero, TIPO_INMUEBLE_INDEX, sobre la relación PROPUESTAS en los atributos ID y TIPO_INMUEBLE, esto con el fin de poder obtener una propuesta por identificador y su tipo de inmueble de una forma más eficiente. De la misma manera, se implementó el índice P_FIE_INDEX sobre la relación de RESERVAS que involucra a los atributos ID_PROPOSTA y FECHA_INICIO_ESTADIA con el propósito de obtener de las reservas, la propuesta sobre la cual se reservó y su fecha en la que inicia la estadía en el inmueble. Estos dos índices son de tipo **B+** en donde el costo es proporcional al número de hojas y la cantidad de operaciones *input-output*. Adicionalmente, los índices TIPO_INMUEBLE_INDEX y P_FIE_INDEX son secundarios, o no integrados, lo que significa que se mantienen en un archivo aparte, al archivo que contiene la relación.

- Sentencia SQL utilizada

```
SELECT PER.*, PP.TIPO
FROM PERSONAS PER
INNER JOIN (

    SELECT R.ID_PERSONA AS PERSO, P.TIPO_INMUEBLE AS TIPO
    FROM RESERVAS R
    INNER JOIN PROPUESTAS P
    ON P.ID = R.ID_PROPOSTA
    WHERE P.ID = 12
    AND R.FECHA_INICIO_ESTADIA BETWEEN '2018-01-18' AND '2018-01-28'

)
PP ON PP.PERSO = PER.ID
ORDER BY PP.TIPO
;
```

- Valores de los parámetros utilizados

Identificador de la oferta de alojamiento sobre la cual se va a hacer el análisis. En este caso, el identificador de la oferta es 12.

Rango de fechas. En este caso es '2018-01-18' AND '2018-01-28'.

- **Plan de consulta**

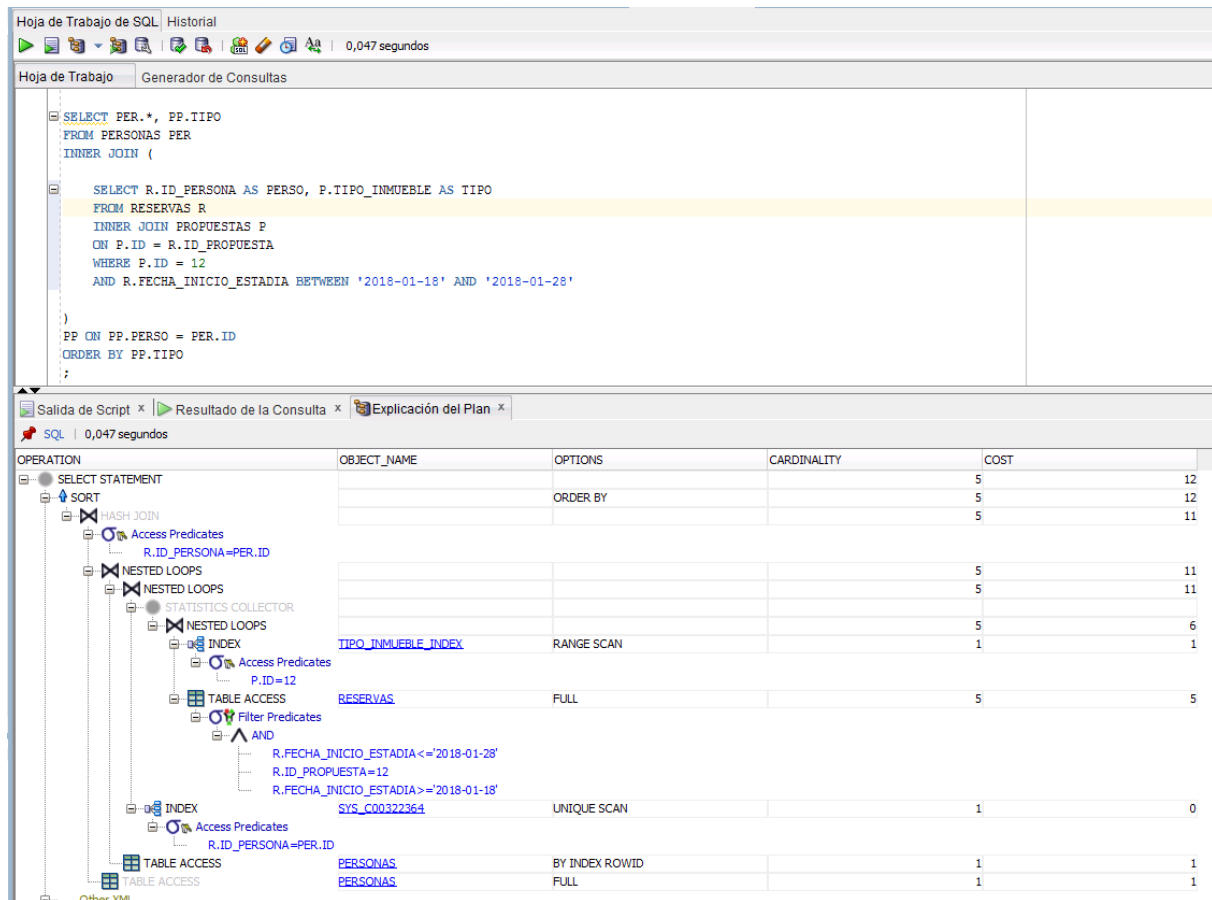


Figura 4. Plan de consulta para el requerimiento RFC 10.

- **Análisis de eficiencia**

Como se observa en sentencia SQL implementada, en primer lugar, se recorren las reservas, juntándolas con las propuestas, que cumplen la condición de tener la fecha de inicio de estadía dentro del rango de fecha establecidas, y con el identificador de la propuesta (oferta de alojamiento) igual al establecido. Luego se obtienen toda la información de las personas cuyos identificadores cuadran con los obtenidos en la anterior consulta.

Por otro lado, el plan de SQL Developer fue primero realizar un *hash join* y luego usando el índice, previamente creado antes de realizar la consulta, se procede a recorrer la relación de RESERVAS para de esta forma consultar las tuplas que satisfacen la condición del rango de fechas y luego se obtienen las personas que realizaron la respectiva reserva.

3.3 RFC 11 Consultar inversa de consumo en Alohandes

- **Selección de índices**

Para este requerimiento se implementaron los mismo dos índices del requerimiento anterior RFC 10.

- **Sentencia SQL utilizada**

```
SELECT PER.*, PP.TIPO
FROM PERSONAS PER
INNER JOIN (
```

```

SELECT R.ID_PERSONA AS PERSO, P.TIPO_INMUEBLE AS TIPO
FROM RESERVAS R
INNER JOIN PROPUESTAS P
ON P.ID = R.ID_PROPUESTA

WHERE P.ID = 12
AND R.FECHA_INICIO_ESTADIA NOT BETWEEN '2018-01-18' AND '2018-01-28'

)
PP ON PP.PERSO = PER.ID
ORDER BY PP.TIPO
;

```

- **Valores de los parámetros utilizados**

Identificador de la oferta de alojamiento sobre la cual se va a hacer el análisis. En este caso, el identificador de la oferta es 12.

Rango de fechas. En este caso es '2018-01-18' AND '2018-01-28'.

- **Plan de consulta**

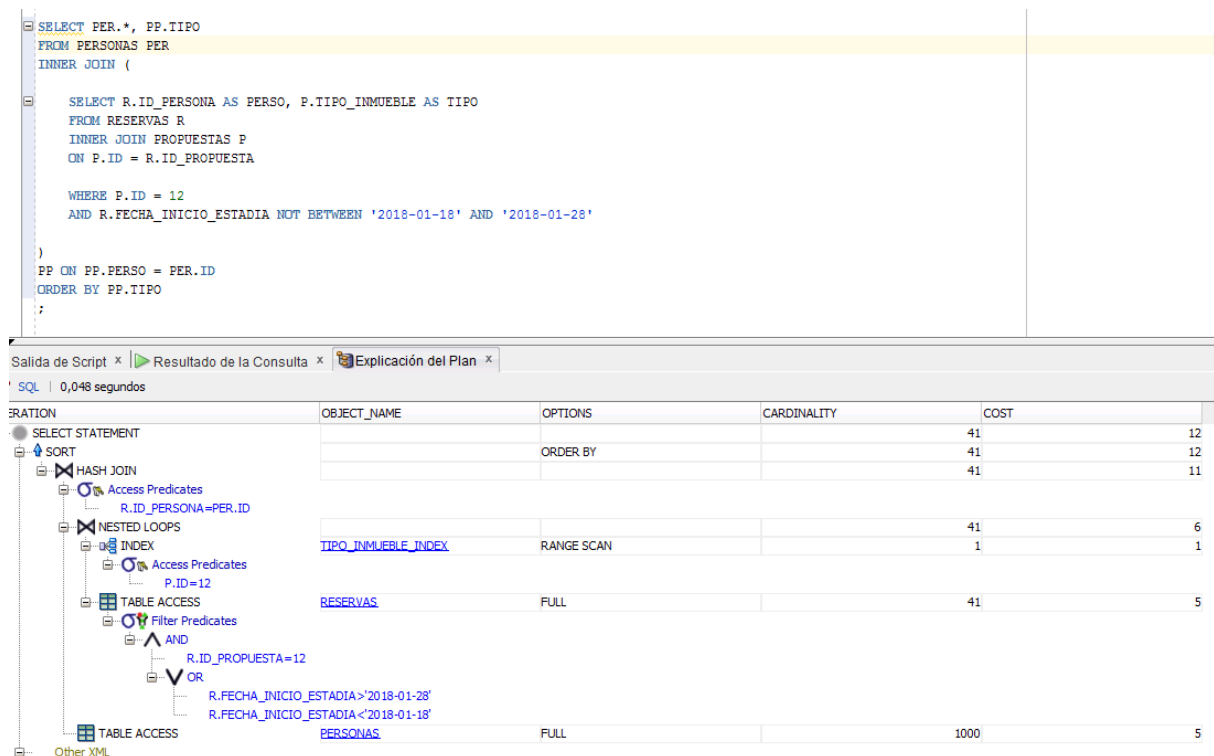


Figura 5. Plan de consulta para el requerimiento RFC 11.

- **Análisis de eficiencia**

De la misma forma en la que se realiza la consulta en el anterior requerimiento de consulta 10, en primer lugar, se recorren las reservas, juntándolas con las propuestas, que cumplen la condición de tener la fecha de inicio de estadía dentro del rango de fecha establecidas, y con el identificador de la propuesta (oferta de alojamiento) igual al establecido. Luego se obtienen toda la información de las personas cuyos identificadores cuadran con los obtenidos en la anterior consulta.

Por otro lado, el plan de SQL Developer fue primero realizar un hash join y luego usando el índice, previamente creado antes de realizar la consulta, se procede a recorrer la relación de

RESERVAS para de esta forma consultar las tuplas que satisfacen la condición del rango de fechas y luego se obtienen las personas que realizaron la respectiva reserva.

3.4 RFC 12 Consultar de funcionamiento

- **Selección de índices**

Para este requerimiento se implementaron Índices en las llaves primarias de las tablas de RESERVAS , PROPUESTAS y USUARIO esto con el fin de facilitar el uso de joins y lograr una mayor tiempo de eficiencia en la consulta. Sin importar la cantidad de datos almacenados Cabe resaltar que el tipo de índices es **B+** en donde el costo es proporcional al número de hojas y la cantidad de operaciones *input-output*.

- **Sentencias sql utilizadas**

```
SELECT to_number(to_char(to_date(R.FECHA_INICIO_ESTADIA, 'YYYY-MM-DD
HH24:MI:SS'), 'WW')) AS "SEMANA", COUNT
(to_number(to_char(to_date(R.FECHA_INICIO_ESTADIA, 'YYYY-MM-DD
HH24:MI:SS'), 'WW')) AS "CANTIDAD RESERVAS"
FROM RESERVAS R
INNER JOIN PROPUESTAS P ON
R.ID_PROPUESTA = P.ID
WHERE upper(P.TIPO_INMUEBLE) = upper('habitacion')
AND ( R.HAY_MULTA IS NULL
OR R.HAY_MULTA = 0 )
GROUP BY to_number(to_char(to_date(R.FECHA_INICIO_ESTADIA, 'YYYY-MM-DD
HH24:MI:SS'), 'WW'))
ORDER BY "CANTIDAD RESERVAS" DESC;
```

```
SELECT W."SEMANA", W."CANTIDAD RESERVAS", PRO.* FROM PROPUESTAS PRO
```

```
INNER JOIN (
```

```
SELECT
```

```
to_number(to_char(to_date(R.FECHA_INICIO_ESTADIA, 'YYYY-MM-DD
HH24:MI:SS'), 'WW')) AS "SEMANA",
COUNT (to_number(to_char(to_date(R.FECHA_INICIO_ESTADIA, 'YYYY-MM-DD
HH24:MI:SS'), 'WW')) AS "CANTIDAD RESERVAS",
r.id_propuesta as "ID Propuesta"
```

```
FROM RESERVAS R
```

```
INNER JOIN PROPUESTAS P ON
R.ID_PROPUESTA = P.ID
```

```
WHERE ( R.HAY_MULTA IS NULL
OR R.HAY_MULTA = 0 )
```

```
GROUP BY to_number(to_char(to_date(R.FECHA_INICIO_ESTADIA, 'YYYY-MM-DD
HH24:MI:SS'), 'WW')), r.id_propuesta
```

```
)
```

```
W
```

```
ON W."ID Propuesta" = pro.id
```

```
ORDER BY "SEMANA", "CANTIDAD RESERVAS" DESC
```

```
;
```

- **Parametros utilizados**

Es necesario indicar si se desea conocer los operadores o los alojamientos mas/ menos populares de la aplicación para esto se debe introducir en el path la palabra alojamiento o la palabra operador, además se debe indicar el ordenamiento haciendo uso de la palabra mas o menos para buscar los alojamientos/ operadores populares

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				11
SORT		ORDER BY		11
HASH		GROUP BY		11
HASH JOIN				66
Access Predicates				
R.ID_PROPOSTA=P.ID				
INDEX	TIPO_INMUEBLE_INDEX	FULL SCAN		8
Filter Predicates				
UPPER(P.TIPO_INMUEBLE)='HABITACION'				
TABLE ACCESS	RESERVAS	FULL		468
Filter Predicates				
OR				
R.HAY_MULTA IS NULL				
R.HAY_MULTA=0				

figura13. Plan de consulta de requerimiento funcional 12

SELECT STATEMENT			136
SORT		ORDER BY	136
MERGE JOIN			136
TABLE ACCESS	PROPUESTAS	BY INDEX ROWID	47
INDEX	SYS_C00322394	FULL SCAN	47
JOIN			136
Access Predicates			
W.ID Propuesta=PRO.ID			
Filter Predicates			
W.ID Propuesta=PRO.ID			
VIEW			
HASH		GROUP BY	136
NESTED LOOPS			136
TABLE ACCESS	RESERVAS	FULL	468
Filter Predicates			468
OR			
R.HAY_MULTA IS NULL			
R.HAY_MULTA=0			
INDEX	SYS_C00322394	UNIQUE SCAN	1
Access Predicates			
R.ID_PROPOSTA=P.ID			

figura14. Plan de consulta de requerimiento funcional 12

• Analisis de Eficiencia

la consulta de este requerimiento consiste, en recorren las propuesta juntándolas con las reservas , que cumplen con ser del mismo operador ,se buscan las propuestas de mayor/menor costo según sea indicado. Finalmente se despliega la información que cumple con las condiciones anteriores

Por otro lado, el plan de SQL Developer fue primero realizar un hash join y luego usando el índice, previamente creado antes de realizar la consulta, se procede a recorrer la relación de propuestas para de esta forma consultar las tuplas que satisfacen la condición de mayor ocupacion.

3.5 RFC 13 Consultar los buenos clientes

• Selección de indices

Para este requerimiento se implementaron Indices en las llaves primarias de las tablas de RESERVAS , PROPUESTAS y USUARIO esto con el fin de facilitar el uso de joins y lograr una mayor tiempo de eficiencia en la consulta. Sin importar la cantidad de datos almacenados, estos indices son de carácter primario y tipo B+ esto hace que las consultas sean proporcionales al numero de hojas y a la cantidad de operaciones *input-output*

• Sentencias SQL utilizadas

```
SELECT
  personas.id AS id1,
  personas.apellido,
  personas.nombre,
```

```

        personas.cedula,
        personas.tipo,
        personas.rol,
        personas.nit,
        personas.email,
        COUNT(reservas.id) AS reservasTotales
FROM
    reservas
    INNER JOIN personas ON personas.id = reservas.id_persona
WHERE
    reservas.COSTO_TOTAL >2600000
GROUP BY
    personas.id,
    personas.apellido,
    personas.nombre,
    personas.cedula,
    personas.tipo,
    personas.rol,
    personas.nit,
    personas.email;

SELECT
    personas.id AS id1,
    personas.apellido,
    personas.nombre,
    personas.cedula,
    personas.tipo,
    personas.rol,
    personas.nit,
    personas.email,
    COUNT(reservas.id) AS reservasTotales
FROM
    reservas
    INNER JOIN personas ON personas.id = reservas.id_persona
WHERE
    reservas.fecha_registro < '2016-12-31'
GROUP BY
    personas.id,
    personas.apellido,
    personas.nombre,
    personas.cedula,
    personas.tipo,
    personas.rol,
    personas.nit,
    personas.email

SELECT
    COUNT(reservas.id) AS reservastotales,
    propuestas.tipo_inmueble,
    hoteles.tipo_habitacion,
    personas.id AS id1,

    personas.nombre,
    personas.apellido,
    personas.cedula,
    personas.tipo,
    personas.rol,
    personas.nit,
    personas.email
FROM
    reservas

```



```

INNER JOIN propuestas ON propuestas.id = reservas.id_propuesta
INNER JOIN hoteles ON hoteles.id = propuestas.id_hotel
INNER JOIN personas ON personas.id = reservas.id_persona
WHERE TIPO_HABITACION='suite'
GROUP BY
propuestas.tipo_inmueble,
hoteles.tipo_habitacion,
personas.id,
personas.nombre,
personas.apellido,
personas.cedula,
personas.tipo,
personas.rol,
personas.nit,
personas.email;

```

- **Parametros Utilizados :**

Para que esta consulta se logre con éxito no es necesario introducir paramteros.

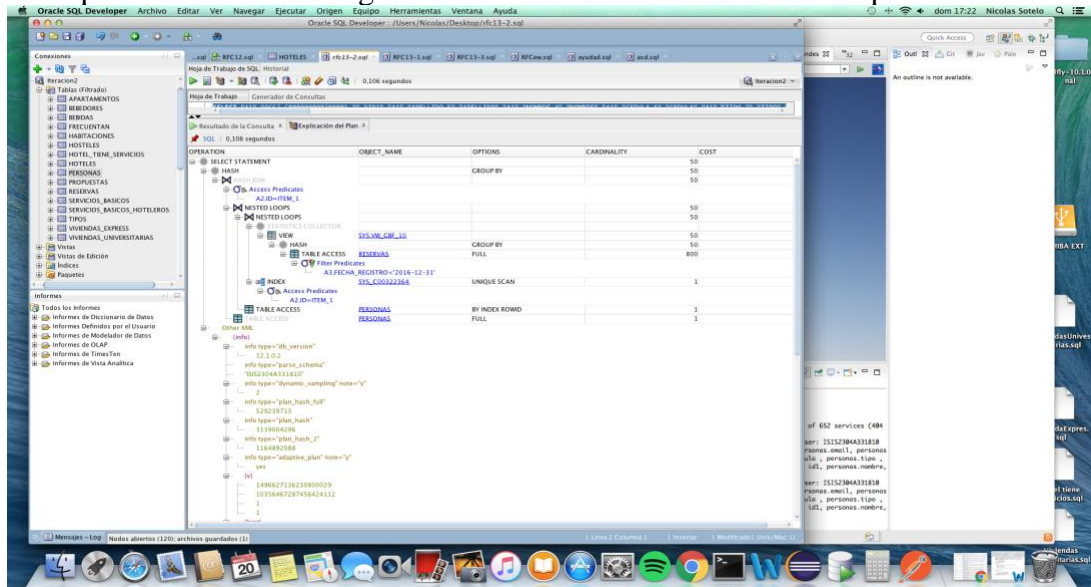


Figura 5. Plan de consulta para el requerimiento RFC 13-1.

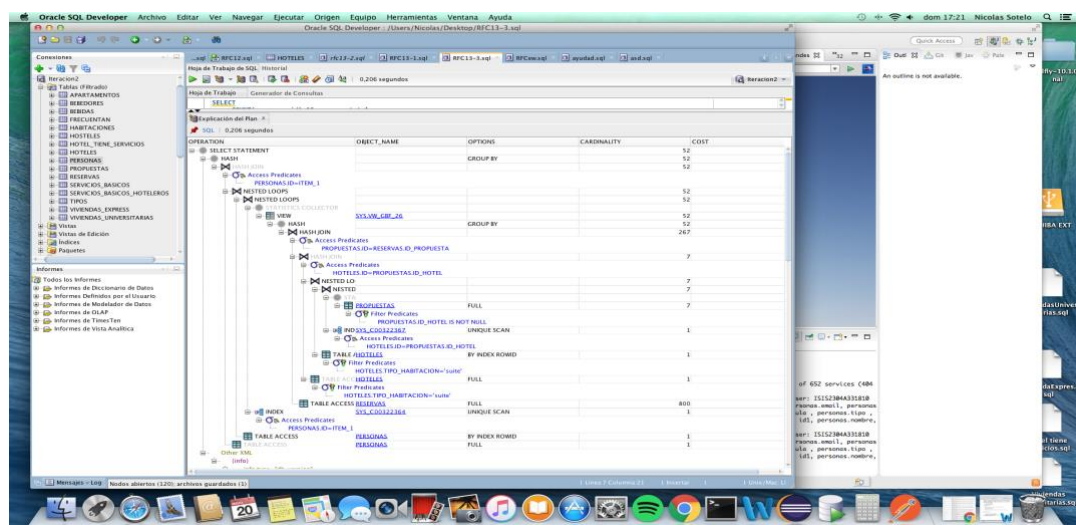


Figura 5. Plan de consulta para el requerimiento RFC 13-2.

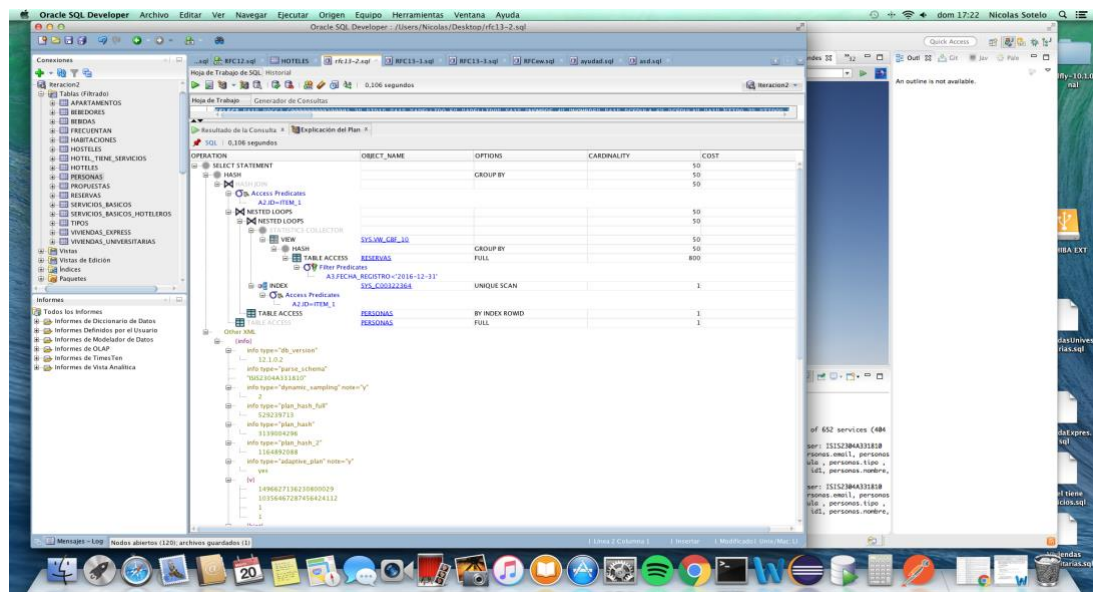


Figura 5. Plan de consulta para el requerimiento RFC 13-3.

- **Análisis De eficiencia**

el plan de SQL Developer realiza en una primera instancia un hash join y luego usando el índice, previamente creado antes de realizar la consulta, se procede a recorrer la relación de RESERVAS para de esta forma se encuentran de manera eficiente las tuplas que permiten diferenciar a un buen cliente

4 Diseño de datos

Se implementó un *loop* en SQL Developer para realizar la carga masiva de datos. Un ejemplo de esto, es con la relación APARTAMENTOS como se muestra a continuación.

```
DECLARE
    ID NUMBER(20) :=1;
    MAXIMO NUMBER(20) := 250000;
    MUEBLE NUMBER(1) := 0;
    COSTO NUMBER(20) := 120;
    CAP NUMBER(20) := 2;
BEGIN
    WHILE ID <= MAXIMO
    LOOP
        INSERT INTO
        APARTAMENTOS( ID, AMOBLADO, COSTO_ADMIN, CAPACIDAD_MAXIMA )
        VALUES ( ID, mueble, COSTO, cap );

        ID := ID + 1;

        IF(MUEBLE = 0 ) THEN
            MUEBLE := 1;
        ELSE
            MUEBLE := 0;
        END IF;

        IF ( COSTO >= 340 ) THEN
            COSTO := COSTO - 97;
```

```

ELSE
    COSTO := COSTO + 26;
END IF;

IF ( CAP >= 20 ) THEN
    CAP := CAP - 3;
ELSE
    CAP := CAP + 1;
END IF;

END LOOP;
END;

```

En la anterior sentencia se empiezan declarando variables que sirven para generar valores distintos en cada atributo y de esta forma evitar la repetición de datos. Luego se implementa la función `WHILE LOOP` en donde en cada recorrido se inserta una nueva tupla en la relación y luego se modifican algunos valores para evitar la repetición de los mismos en distintas tuplas. Un ejemplo de lo anterior es incrementar en 1 el valor del identificador.

Para las demás relaciones en la base de datos se implementó la misma metodología. Las sentencias completas para cada relación se pueden encontrar en la carpeta *insert* que se encuentra en docs > Iteracion 3 > SQL.

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

Aunque las consultas delegadas a un manejador de bases de datos como SQLDeveloper y la ejecución del requerimiento en Java pueden llevar a las mismas soluciones, es importante analizar la diferencia que ambos procesos tienen entre sí. Principalmente, la diferencia entre un manejador de bases de datos y Java surge en la complejidad que existe a la hora de procesar sentencias SQL.

Por su parte Java requiere de hacer comparaciones entre tablas, filtrar información, hacer uso de sentencias como `while` e `if` para evaluar condiciones de búsqueda o encontrar información que se encuentra entre distintas tablas. Además se debe cargar la información en distintas estructuras de datos lo que puede consumir bastante memoria y tiempo de ejecución. Por el contrario los manejadores de bases de datos tienen acceso directo a la información que se quiere procesar y cuentan con distintas funciones y herramientas, como es el caso de los índices en cada tabla, que agilizan el proceso de recolección de datos, búsquedas y modificaciones.

6 Escenarios de prueba

6.1 RFC 10 y 11 Consumo Alohandes

En este requerimiento se solicitan varios parámetros. Primero se especifica si el usuario que hace la consulta es cliente o administrador. Si es administrador, el primer atributo tiene como valor 1, haciendo referencia a la representación en binario de *true*. Luego se indica el identificador de la oferta de alojamiento que se piensa analizar. Luego se indica un rango de fechas en formato YYYY-MM-DD. Seguido por el tipo de ordenamiento en el que se desea obtener el resultado, ya sea por tipo de inmueble o por el identificador de la persona (`id_persona`). Luego, si el usuario es un cliente, se especifica su identificador (cero si es administrador). Y por último se especifica si se desea obtener el requerimiento funcional de consulta 10 (con la palabra “normal”) o si se desea obtener su versión inversa que es el requerimiento funcional de consulta 11 (con la palabra “inverso”). El resultado se observa en

la siguiente figura una lista de usuarios en donde el usuario que realiza la consulta es un administrador, el identificador de la oferta a analizar es 12, el rango de fechas es desde 2018-01-18 hasta 2018-01-28, se desea obtener el resultado ordenado por tipo de inmueble, como es un administrador se pone 0 en el campo en donde va el identificador del cliente, y finalmente se especifica cual requerimiento se quiere obtener.

```
GET http://localhost:8080/Alohandes_IT1/rest/consultas/1/12/2018-01-18/2018-01-28/inmuelle/0/normal

Pretty Raw Preview JSON

[
  {
    "id": 10,
    "nombre": "Pepi",
    "apellido": "Poston",
    "tipo": "empresa",
    "rol": "cliente",
    "nit": null,
    "cedula": "16326430",
    "email": "pposton9@cnn.com",
    "propuestas": [],
    "costo_multa": 0,
    "inmuelle": "Habitacion"
  },
  {
    "id": 36,
    "nombre": "Juan",
    "apellido": "Ravens",
    "tipo": "profesor",
    "rol": "cliente",
    "nit": null,
    "cedula": "58775148",
    "email": "jgravensz@geocities.com",
    "propuestas": [],
    "costo_multa": 0,
    "inmuelle": "Habitacion"
  }
]
```

Figura 8. Escenario de purbea requerimiento de consulta 10 como administrador.

```
GET http://localhost:8080/Alohandes_IT1/rest/consultas/0/13/2018-01-18/2018-01-28/inmuelle/14/normal

Pretty Raw Preview JSON

[
  {
    "id": 14,
    "nombre": "Cristy",
    "apellido": "Saltmarsh",
    "tipo": "empresa",
    "rol": "operador",
    "nit": null,
    "cedula": "22857002",
    "email": "csaltmarsh@tinypic.com",
    "propuestas": [],
    "costo_multa": 0,
    "inmuelle": "Hotel"
  }
]
```

Figura 9. Escenario de purbea requerimiento de consulta 10 como cliente.

```
GET http://localhost:8080/Alohandes_IT1/rest/consultas/1/12/2018-01-18/2018-01-28/inmueble/0/inverso

Pretty Raw Preview JSON

1 [
2   {
3     "id": 3,
4     "nombre": "Kimbell",
5     "apellido": "Hargreves",
6     "tipo": "invitado",
7     "rol": "cliente",
8     "nit": null,
9     "cedula": "4897929",
10    "email": "khargreves2@sitemeter.com",
11    "propuestas": [],
12    "costo_multa": 0,
13    "inmueble": "Habitacion"
14  },
15  {
16    "id": 50,
17    "nombre": "Mylo",
18    "apellido": "Bewshea",
19    "tipo": "empleado",
20    "rol": "cliente",
21    "nit": null,
22    "cedula": "81632150",
23    "email": "mbewsheald@yahoo.co.jp",
24    "propuestas": [],
25    "costo_multa": 0,
26    "inmueble": "Habitacion"
27  }
28 ]
```

Figura 10. Escenario de purbea requerimiento de consulta 11 como administrador.

```
GET http://localhost:8080/Alohandes_IT1/rest/consultas/0/13/2018-01-18/2018-01-28/inmueble/14/inverso

Pretty Raw Preview JSON

1 [
2   {
3     "id": 14,
4     "nombre": "Cristy",
5     "apellido": "Saltmarshe",
6     "tipo": "empresa",
7     "rol": "operador",
8     "nit": null,
9     "cedula": "22857002",
10    "email": "csaltmarshed@tinypic.com",
11    "propuestas": [],
12    "costo_multa": 0,
13    "inmueble": "Hotel"
14  },
15  {
16    "id": 14,
17    "nombre": "Cristy",
18    "apellido": "Saltmarshe",
19    "tipo": "empresa",
20    "rol": "operador",
21    "nit": null,
22    "cedula": "22857002",
23    "email": "csaltmarshed@tinypic.com",
24    "propuestas": [],
25    "costo_multa": 0,
26    "inmueble": "Hotel"
27  }
28 ]
```

Figura 10. Escenario de purbea requerimiento de consulta 11 como cliente.


6.2 RFC 12 Mayor menor demanda y operadores más y menos solicitados

GET ▾

http://localhost:8080/Alohandes_IT1/rest/consultas/algo

PrettyRawPreview

JSON ▾



```
2  {
3    "id": 10,
4    "tipo_inmueble": "Aapartamento",
5    "capacidad_maxima": 4,
6    "cantidad_dias_disponibles": 21,
7    "fecha_final_disponibilidad": "5/19/2015",
8    "fecha_inicio_disponibilidad": "7/14/2009",
9    "disponible": false,
10   "sub_total": 0,
11   "vivienda_universitaria": null,
12   "se_va_retirar": true,
13   "disonible": false,
14   "id_persona": 322,
15   "apartamento": null,
16   "habitacion": null,
17   "hostel": null,
18   "hotel": null,
19   "seVaRetirar": true,
20   "vivienda_express": null,
21   "vivienda_universitarias": null
22 },
23 {
24   "id": 10,
25   "tipo_inmueble": "Aapartamento",
26   "capacidad_maxima": 4,
27   "cantidad_dias_disponibles": 21,
28   "fecha_final_disponibilidad": "5/19/2015",
29   "fecha_inicio_disponibilidad": "7/14/2009",
30   "disponible": false,
31   "sub_total": 0,
32   "vivienda_universitaria": null,
33   "se_va_retirar": true,
```

Figura 10. Operadores mas solicitados.

```
GET http://localhost:8080/Alohandes_IT1/rest/consultas/mayor/semana/habitacion

Pretty Raw Preview JSON

{
  "tiempo": "8",
  "resultado": 10,
  "_resultado": 10
},
{
  "tiempo": "6",
  "resultado": 8,
  "_resultado": 8
},
{
  "tiempo": "10",
  "resultado": 8,
  "_resultado": 8
}
```

Figura 10.1. Ofertas con mayor demanda por semana.

```
GET http://localhost:8080/Alohandes_IT1/rest/consultas/menor/semana/habitacion

Pretty Raw Preview JSON

{
  "tiempo": "5",
  "resultado": 2,
  "_resultado": 2
},
{
  "tiempo": "11",
  "resultado": 2,
  "_resultado": 2
},
{
  "tiempo": "4",
  "resultado": 2,
  "_resultado": 2
}
```

Figura 10.2. Ofertas con menor demanda por semana.

6.3 RFC 13 Buenos Clientes

Para este requerimiento no son necesarios ningunos datos solo es necesario acceder al path *"/consultas/buenos clientes"*

```
1 {
2   "concurrentes": [{}],
704   "costosos": [{}],
916   "suites": [{}],
1506 }
```

figura11. EscenarioDe prueba Requerimiento de consulta 13

7 Consideraciones

En los documentos anexos se encuentran la colección de pruebas Postman (`docs > Iteracion 3 > Pruebas`).

En la carpeta de SQL (`docs > Iteracion 3 > SQL`) se encuentran las sentencias SQL implementadas para: crear las tablas y sus atributos, poblar las tablas, realizar los requerimientos funcionales, y se encuentran las sentencias utilizadas para completar los requerimientos funcionales de consulta.

8 Resultados

Con lo observado en la presente iteración, se concluye que un índice, al ser una estructura de datos que mantiene en orden el resumen de la información almacenada en una base de datos, se facilitan y agilizan las consultas que se hacen sobre dicha base de datos. En esta iteración se implementaron índices compuesto, en dónde se involucran varios atributos de una relación, para obtener de forma más rápida tuplas que cumplieran con los parámetros establecidos, como por ejemplo el tipo de inmueble, que se establecían en cada requerimiento de consulta. De la misma manera, se implementaron índices de tipo B+ puesto a su rendimiento al realizar consultas que involucran valores dentro de un rango o llaves parciales, como lo era en el caso de los requerimientos presentes. Con todo esto se logró la meta de realizar consultas en la base de datos en menos de 0.8 segundos. Vale la pena aclarar que por cada relación de inmueble (que en total suman 6 relaciones distintas) se insertaron 250 000 tuplas en cada una; y además en la relación `PERSONAS` se insertaron 800 000 tuplas para garantizar un volumen de datos tan grande que no cupieran en la memoria principal.