

**Universidad De Los Andes**  
Sistemas Transaccionales

**Santiago Fajardo**  
201813423

**Nicolas Esteban Cárdenas**  
201813011

**Iteración 3**

## Índices

La selección de los índices se realizó a partir de la selectividad de los atributos que se requerían en las consultas 9 a 12.

### RFC 9

Para esta consulta, se consideró usar un árbol B+, dado que la respuesta debe ser dada en rangos (Se solicitan varios servicios, ips, etc...). Igualmente, se consideró también hacer un índice primario sobre Paciente.Id\_usuario, Usuario.id, Ips.id ya que su selectividad es igual o menor a 25%. También se crearon índices secundarios sobre los campos:

- Consulta.fecha
- Usuario.nombre, Usuario.fechaNacimiento
- Ips.nombre, Ips.localizacion

dado que su selectividad es inferior al 25%, y son empleados en la consulta.

### RFC 10

Para la consulta, se consideró usar también un árbol B+, dado que se requieren también consultas en rangos. Para los índices, se considera crear índices primarios para los campos:

- Ips\_Servicios.id\_servicio
- Ips\_Servicios.id\_ips
- Ips.Id

Dado que su selectividad lo amerita. Por otra parte, se consideró también crear índices secundarios para los campos por la misma razón anterior:

- Consulta.Id\_Orden
- Consulta.Cumplida

### RFC 11

Para la consulta se considera el uso de un árbol B+, debido a que se requiere consultar datos específicos en un rango de fechas, la diferencia es que en esta consulta se busca agrupar todo por semanas. Para los índices, se consideró los siguientes índices primarios:

- Ips.Id
- Servicio.Id
- Paciente.Id\_Usuario

En esta consulta, no se tiene en cuenta el uso de índices secundarios, pues solo se requiere hacer join mediante el uso de las llaves primarias de las tablas sobre las que se quiere realizar la consulta. La selección de índices se da gracias a la selectividad de estos con respecto a la densidad de datos

## RFC 12

Para esta consulta se considera el uso de una estructura como árbol B+, pues lo principal es filtrar por una dada característica a los usuarios. Los índices primarios planteados para cumplir esta consulta serían los siguientes:

- Paciente.Id\_Usuario
- Orden.Id
- Servicio.Id

Por otro lado, los índices secundarios planteados para poder llevar acabo esta consulta son los siguientes:

- Ordenes\_Servicios.Id\_Orden
- Ordenes\_Servicios.Id\_Servicio

De esta manera se asegura que se cumpla la eficiencia al momento de realizar la consulta, pues siempre se indexa por los criterios bajo los cuales se desea realizar la consulta

## INDICES ESCOGIDO POR ORACLE

OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME
1 ISIS2304C041920	CAMPANA_PK	NORMAL	ISIS2304C041920	CAMPANA
2 ISIS2304C041920	CH_PK	NORMAL	ISIS2304C041920	CAMPANA_HORARIO
3 ISIS2304C041920	CONSULTA_PK	NORMAL	ISIS2304C041920	CONSULTA
4 ISIS2304C041920	SYS_C00701481	NORMAL	ISIS2304C041920	CONSULTA
5 ISIS2304C041920	EPS_PK	NORMAL	ISIS2304C041920	EPS
6 ISIS2304C041920	HORARIO_PK	NORMAL	ISIS2304C041920	HORARIO
7 ISIS2304C041920	IPS_PK	NORMAL	ISIS2304C041920	IPS
8 ISIS2304C041920	IH_PK	NORMAL	ISIS2304C041920	IPS_HORARIO
9 ISIS2304C041920	IM_M	NORMAL	ISIS2304C041920	IPS_MEDICOS
10 ISIS2304C041920	IS_PK	NORMAL	ISIS2304C041920	IPS_SERVICIOS
11 ISIS2304C041920	MEDICAMENTO_PK	NORMAL	ISIS2304C041920	MEDICAMENTO
12 ISIS2304C041920	MEDICO_PK	NORMAL	ISIS2304C041920	MEDICO
13 ISIS2304C041920	SYS_C00701446	NORMAL	ISIS2304C041920	MEDICO
14 ISIS2304C041920	MED_PAC_PK	NORMAL	ISIS2304C041920	MEDICOS_PACIENTES
15 ISIS2304C041920	ORDEN_PK	NORMAL	ISIS2304C041920	ORDEN
16 ISIS2304C041920	ORDMED_PK	NORMAL	ISIS2304C041920	ORDENES_MEDICAMENTOS
17 ISIS2304C041920	ORDENES_SERVICIOS_PK	NORMAL	ISIS2304C041920	ORDENES_SERVICIOS
18 ISIS2304C041920	PACIENTE_PK	NORMAL	ISIS2304C041920	PACIENTE
19 ISIS2304C041920	SERVICIO_PK	NORMAL	ISIS2304C041920	SERVICIO
20 ISIS2304C041920	SH_PK	NORMAL	ISIS2304C041920	SERVICIO_HORARIO
21 ISIS2304C041920	SYS_C00701471	NORMAL	ISIS2304C041920	URGENCIA
22 ISIS2304C041920	URGENCIA_PK	NORMAL	ISIS2304C041920	URGENCIA
23 ISIS2304C041920	SYS_C00701469	NORMAL	ISIS2304C041920	URGENCIA
24 ISIS2304C041920	SYS_C00701470	NORMAL	ISIS2304C041920	URGENCIA
25 ISIS2304C041920	SYS_C00701440	NORMAL	ISIS2304C041920	USUARIO
26 ISIS2304C041920	SYS_C00701439	NORMAL	ISIS2304C041920	USUARIO
27 ISIS2304C041920	USUARIO_PK	NORMAL	ISIS2304C041920	USUARIO

Los índices mostrados anteriormente, fueron los índices automáticamente generados por el SMBD, ya que los campos sobre los cuales se crearon los índices (secundarios) son de tipo UNIQUE. Por otra parte, para las llaves primarias se crearon también índices primarios únicos. Sí ayudan a hacer más eficientes los requerimientos funcionales ya que permiten un acceso más rápido a los campos de las tablas.

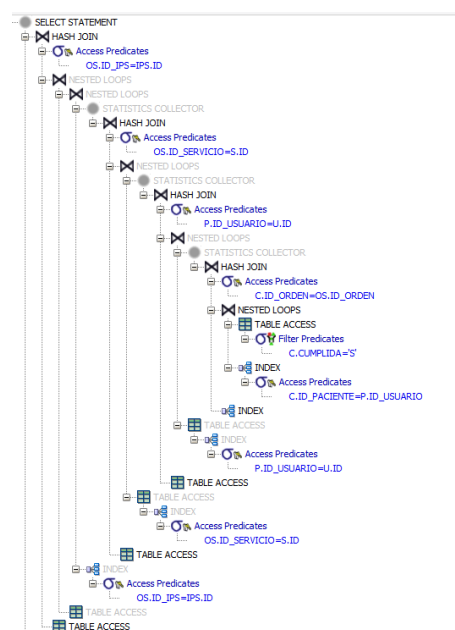
## PLANES DE CONSULTA

### RFC 9

#### Propuesta:

Para este requerimiento de consulta, se consideró el método de joins de Hash, ya que la información de las tablas no cabe en memoria principal, y el método es útil para equi-joins.

#### Explain Plan Oracle:



SELECT STATEMENT			4	14
HASH JOIN			4	14
Access Predicates				
OS.ID_IPS=IPS.ID				
NESTED LOOPS			4	14
NESTED LOOPS				
STATISTICS COLLECTOR				
HASH JOIN			4	11
Access Predicates				
OS.ID_SERVICIO=S.ID				
NESTED LOOPS			4	11
STATISTICS COLLECTOR				
HASH JOIN			4	8
Access Predicates				
P.ID_USUARIO=U.ID				
NESTED LOOPS			4	8
STATISTICS COLLECTOR				
HASH JOIN			4	5
Access Predicates				
C.ID_ORDEN=OS.ID_ORDEN				
NESTED LOOPS			4	3
TABLE ACCESS	CONSULTA	FULL	4	3
Filter Predicates				
C.CUMPLIDA='S'				
INDEX	PACIENTE_PK	UNIQUE SCAN	1	0
Access Predicates				
C.ID_PACIENTE=P.ID_USUARIO				
INDEX	ORDENES_SERVICIOS_PK	FAST FULL SCAN	10	2
TABLE ACCESS	USUARIO	BY INDEX ROWID	1	3
Access Predicates	USUARIO_PK	UNIQUE SCAN		
P.ID_USUARIO=U.ID				
TABLE ACCESS	USUARIO	FULL	10	3
INDEX	SERVICIO	BY INDEX ROWID	1	3
Access Predicates	SERVICIO_PK	UNIQUE SCAN		
OS.ID_SERVICIO=S.ID				
TABLE ACCESS	SERVICIO	FULL	10	3
INDEX	IPS_PK	UNIQUE SCAN		
OS.ID_IPS=IPS.ID				
TABLE ACCESS	IPS	BY INDEX ROWID	1	3
TABLE ACCESS	IPS	FULL	10	3

Se puede apreciar que para todos los joins se usó el método hash join, igual al que fue propuesto.

### RFC 10

Se consideró también el método de joins de Hash, pues las tablas no caben en memoria primaria y se requieren hacer equi-joins.

SELECT STATEMENT				16	19
HASH JOIN				16	19
Access Predicates					
SERV.ID_SERVICIO=SER.ID					
HASH JOIN				10	8
Access Predicates					
SER.ID=IPSSERV.ID_SERVICIO					
NESTED LOOPS				10	8
STATISTICS COLLECTOR					
HASH JOIN				10	5
Access Predicates					
IPSSERV.ID_IPS=IPS.ID					
NESTED LOOPS				10	5
STATISTICS COLLECTOR					
INDEX	IS_PK	FAST FULL SCAN		10	2
TABLE ACCESS	IPS	BY INDEX ROWID		1	3
INDEX	IPS_PK	UNIQUE SCAN			
Access Predicates					
IPSSERV.ID_IPS=IPS.ID					
TABLE ACCESS	IPS	FULL		10	3
INDEX	SERVICIO_PK	UNIQUE SCAN			
Access Predicates					
SER.ID=IPSSERV.ID_SERVICIO					
TABLE ACCESS	SERVICIO	BY INDEX ROWID		1	3
TABLE ACCESS	SERVICIO	FULL		10	3
VIEW				16	11
SORT		UNIQUE		16	11
UNION-ALL				16	11
HASH JOIN				6	5
Access Predicates					
B.ID_ORDEN=C.ID_ORDEN					
NESTED LOOPS				6	5
STATISTICS COLLECTOR					
TABLE ACCESS	CONSULTA	FULL		6	3
Filter Predicates					
C.CUMPLIDA=N					
INDEX	ORDENES_SERVICIOS_PK	RANGE SCAN		1	2
Access Predicates					
B.ID_ORDEN=C.ID_ORDEN					
INDEX	ORDENES_SERVICIOS_PK	FAST FULL SCAN		10	2
HASH JOIN		ANTI		10	4
Access Predicates					
S.ID=ID_SERVICIO					

Se puede apreciar que el plan de consulta también consiste en hash joins.

## RFC 11

Se propone usar un Hash Join, pues muchas de las condiciones para que se cumpla la consulta es usando EquiJoins. A pesar de que se requiere agrupar por un rango de semanas, nunca es requerido como una condición del where, por lo cual no es necesario tenerlo en cuenta para la selección del tipo de Join.

Oracle indica lo mismo en el plan de consulta que arroja

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	
SELECT STATEMENT				87	12
SORT		ORDER BY		87	12
HASH		GROUP BY		87	12
VIEW				87	10
HASH		GROUP BY		87	10
HASH JOIN				87	9
Access Predicates OS.ID_IPS=I.ID					
HASH JOIN				87	6
Access Predicates OS.ID_ORDEN=O.ID					
TABLE ACCESS ORDENES_SERVICIOS		FULL		87	3
TABLE ACCESS ORDEN		FULL		101	3
TABLE ACCESS IPS		FULL		100	3

## RFC 12

Para esta consulta se propone usar un Hash Join, pues se desea buscar aquellos afiliados que cumplan con ciertas condiciones. No es necesario de una búsqueda en rangos , por lo tanto usar otro tipo de Joins no sería lo más útil. El plan de consulta presentado es el siguiente:

## SENTENCIAS SQL

--RFC9

```
SELECT u.*, s.nombre, c.fecha, ips.nombre
FROM Paciente p, Consulta c, Ordenes_Servicios os, Usuario u, Servicio s, Ips ips
WHERE
    c.id_paciente = p.id_usuario
    AND p.id_usuario = u.id
    AND c.cumplida = 'S'
    AND c.id_orden = os.id_orden
    AND os.id_servicio = s.id
    AND os.id_ips = ips.id
```

--RFC10

--Selecciona los nombres de servicios que no fueron prestados y sus ips que no los prestaron

```
SELECT ser.nombre as servicio, ips.nombre as ips
FROM (
    -- Servicios no reservados + servicios reservados y no cumplidos
    SELECT a.id_servicio as id_servicio
    FROM ( -- Servicios reservados pero no cumplidos
        SELECT b.id_servicio as id_servicio
        FROM Ordenes_Servicios b, Consulta c
        WHERE b.id_orden = c.id_orden
        AND c.cumplida = 'N'
    ) a
    UNION
    SELECT b.id_servicio
    FROM ( -- Servicios que niquiera fueron reservados
        SELECT s.id as id_servicio
        FROM Servicio s
        WHERE s.id NOT IN (SELECT id_servicio FROM Ordenes_Servicios)
    ) b
) servs, Ips_Servicios ipsServ, IPS ips, Servicio ser
WHERE servs.id_servicio = ser.id
    AND ser.id = ipsServ.id_servicio
    AND ipsServ.id_ips = ips.id;
```

--Selecciona las fechas en las que los servicios no fueron utilizados

```
SELECT c.fecha
FROM Consulta c
WHERE c.cumplida = 'N';
```

--Selecciona los afiliados que no utilizaron servicios

```
SELECT u.nombre
FROM Usuario u, Paciente p, Consulta c
WHERE p.id_usuario = u.id
    AND p.id_usuario NOT IN (SELECT id_paciente FROM Consulta);
```

```

114 --RFC11
115
116 --Servicio Mas Consumido por Semana
117 select Año, Semana, max(cant)
118 from (
119     select s.nombre, to_char(o.fecha - 7/24,'IYYY') Año , to_char(o.fecha - 7/24,'IW') Semana ,count(s.id) cant
120     from ordenes_servicios os, servicio s, orden o
121     where os.id_servicio = s.id
122     and os.id_orden = o.id
123     group by s.nombre, o.fecha, s.id, to_char(o.fecha - 7/24,'IYYY'), to_char(o.fecha- 7/24,'IW')
124     order by o.fecha
125 ) m
126 group by Año, Semana
127 order by Semana;
128
129 --Servicio Menos Consumido por Semana
130 select Año, Semana, max(cant)
131 from (
132     select s.nombre, to_char(o.fecha - 7/24,'IYYY') Año , to_char(o.fecha - 7/24,'IW') Semana ,count(s.id) cant
133     from ordenes_servicios os, servicio s, orden o
134     where os.id_servicio = s.id
135     and os.id_orden = o.id
136     group by s.nombre, o.fecha, s.id, to_char(o.fecha - 7/24,'IYYY'), to_char(o.fecha- 7/24,'IW')
137     order by o.fecha
138 ) m
139 group by Año, Semana
140 order by Semana;
141
142 --Ips Mas Solicitada
143 select Año, Semana, max(cant)
144 from (
145     select i.nombre IPS, to_char(o.fecha - 7/24,'IYYY') Año , to_char(o.fecha - 7/24,'IW') Semana ,count(os.id_ips) cant
146     from ordenes_servicios os, servicio s, orden o, IPS i
147     where os.id_servicio = s.id
148     and os.id_orden = o.id
149     and os.id_ips = i.id
150     group by i.nombre, o.fecha, s.id, to_char(o.fecha - 7/24,'IYYY'), to_char(o.fecha- 7/24,'IW')
151     order by o.fecha
152 ) m
153 group by Año, Semana
154 order by Semana;
155
156 --Ips Menos Solicitada
157 select Año, Semana, min(cant)
158 from (
159     select i.nombre IPS, to_char(o.fecha - 7/24,'IYYY') Año , to_char(o.fecha - 7/24,'IW') Semana ,count(os.id_ips) cant
160     from ordenes_servicios os, servicio s, orden o, IPS i
161     where os.id_servicio = s.id
162     and os.id_orden = o.id
163     and os.id_ips = i.id
164     group by i.nombre, o.fecha, s.id, to_char(o.fecha - 7/24,'IYYY'), to_char(o.fecha- 7/24,'IW')
165     order by o.fecha
166 ) m
167 group by Año, Semana
168 order by Semana;
169

```

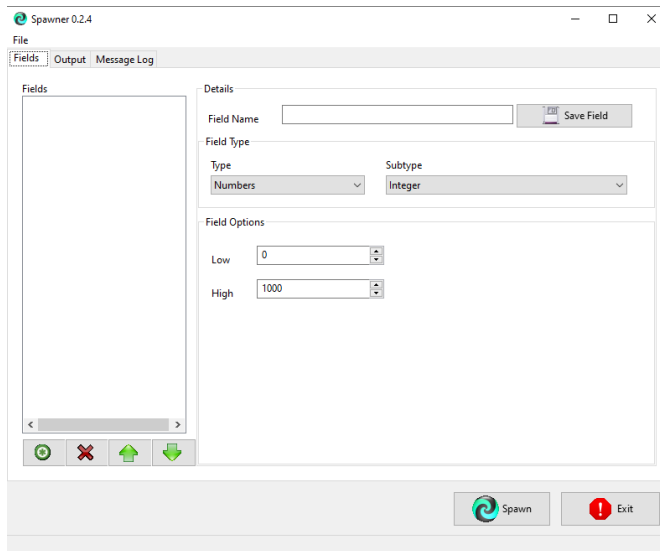
## CARGA DE DATOS

Para este proceso se utilizó un programa llamado spawner.exe, el cual permite crear datos aleatorios de cualquier cantidad para poblar las tablas que se deseen. En este caso, la distribución de datos se realizó de la siguiente forma

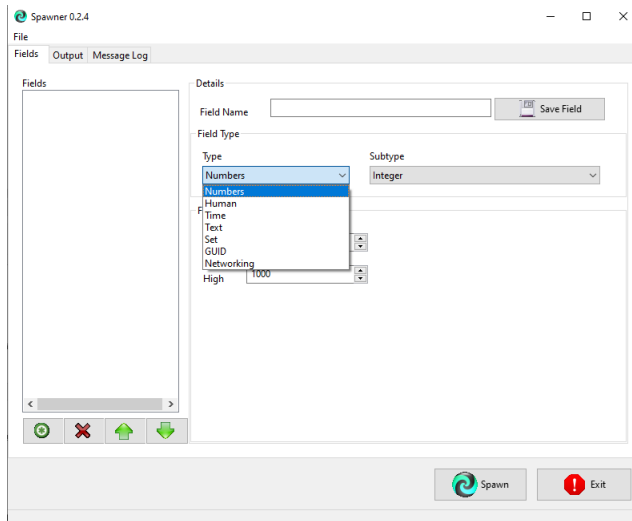
<b>Total datos</b>	815.401
<b>Usuarios</b>	200.000
<b>Servicios</b>	400
<b>Pacientes</b>	150.000

<b>Ordenes_Servicios</b>	100.000
<b>Ordenes</b>	100.000
<b>Médicos</b>	50.000
<b>IPS</b>	15.000
<b>EPS</b>	1
<b>Consultas</b>	200.000

El programa spawner.exe se ve de la siguiente manera



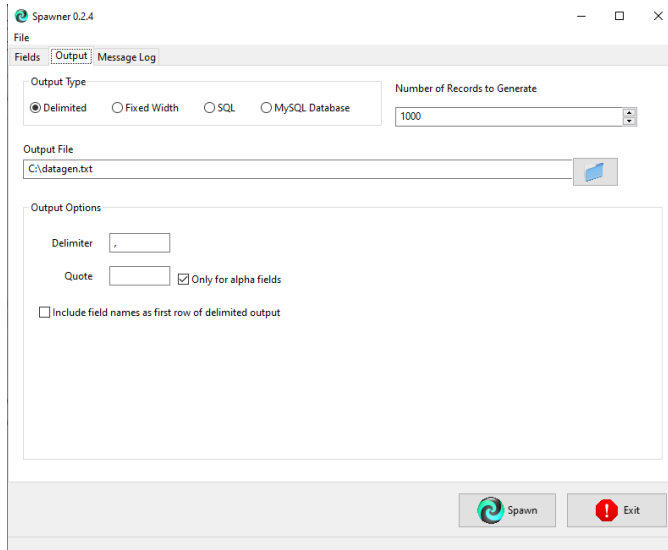
Donde el botón circular verde de la parte inferior izquierda permite el ingreso de campos, se pueden escoger cuantos campos se desee. Adicional a esto, en la ventana Field Type, se puede escoger distintos tipos de datos, como se muestra a continuación



Y se puede escoger un SubType, donde en esta opción se puede escoger el patrón de como se crean lo datos y muchas otras opciones.

Una vez se escogen los campos que se desean crear, en la ventana Output se escoge el numero de tuplas que se desea crear





Al realizar este proceso, se genera un archivo CSV, el cual puede ser cargado desde SQL Developer dando click derecho sobre la tabla a la cual se desea agregar los datos y dando en la opción importar datos.