

Iteración 4 - PuertoAndes

Diseño físico y Optimización de consultas

José Daniel Fandiño, Juan José Castro
Sistemas Transaccionales 2016-1
Universidad de los Andes, Bogotá, Colombia
{jj.castro10, jd.fandino10}@uniandes.edu.co
Fecha de presentación: Abril 28 de 2016

Iteración 4 - PuertoAndes

Diseño físico y Optimización de consultas

- 1 Documentación del diseño físico
 - 1.1 Selección de índices
 - 1.2 Índices creados de forma automática
- 2 Documentación del análisis por requerimiento
 - 2.1 RFC7
 - 2.1.1 Escenario de pruebas
 - 2.1.2 Eficiencia
 - 2.2 RFC8
 - 2.2.1 Escenario de pruebas
 - 2.2.2 Eficiencia
 - 2.3 RFC9
 - 2.3.1 Escenario de pruebas
 - 2.3.2 Eficiencia
 - 2.4 RFC10
 - 2.4.1 Escenario de pruebas
 - 2.4.2 Eficiencia
- 3 Carga de Tablas
- 4 Comparación de ejecución de consultas

1 Documentación del diseño físico

Justifique la selección de índices desde el punto de vista de cada uno de los requerimientos funcionales. Indique claramente cuál es el tipo de índice utilizado (B+, Hash, ..., primario, secundario) y tenga en cuenta el costo de almacenamiento y mantenimiento asociado a los índices.

1.1 Selección de índices

Requerimiento	Tabla	Índice	Tipo índice	Justificación
RFC7 y RFC8	Arribos	ID_BUQUE,	Índice HASH	Utilizando este índice se puede hacer join con respecto al id_buque de manera más eficiente. El índice hash es el más apropiado en este caso, pues permite búsquedas y joins muy rápidos según relaciones de igualdad.
		FECHA	Índice B+	Con este índice se puede filtrar de manera más eficiente el rango de fechas. La elección del tipo de índice se debe a la alta selectividad y cardinalidad de los datos, los cuales se filtran según relaciones de mayor o menor (>, <, etc)
	Buques	Nombre	Índice HASH	El índice hash es el más apropiado en este caso, pues permite búsquedas y joins muy rápidos según relaciones de igualdad
		TipoBuque	Índice BITMAP	Ya que TipoBuque tiene un número muy limitado de valores, un índice bitmap ofrece mejor desempeño debido a la baja cardinalidad
	TipoCarga	TipoCarga, IdBuque	Índice BITMAP	Ya que TipoCarga tiene un número muy limitado de valores, un índice bitmap ofrece mejor desempeño debido a la baja cardinalidad.
	Carga	Valor	Índice B+	Ya que de esta manera es más eficiente realizar búsquedas sobre este atributo.
RFC9 y RFC10		id_exportador		

		Tipo	Índice BITMAP	Ya que Tipo tiene un número muy limitado de valores, un índice bitmap ofrece mejor desempeño debido a la baja cardinalidad.
	Movimiento Marítimo, Movimiento Terrestre	Id_carga	Índice HASH	El índice hash es el más apropiado en este caso, pues permite búsquedas y joins muy rápidos según relaciones de igualdad.

1.2 Índices creados de forma automática

Arribos:

	INDEX_OWNER	INDEX_NAME	UNIQUENESS	STATUS	INDEX_TYPE	JOIN_INDEX	COLUMNS
1	ISIS2304B061610	ARRIBOS_PK	UNIQUE	VALID	NORMAL	NO	ID_BUQUE, FECHA, ID_MUELLE

Salidas:

	INDEX_OWNER	INDEX_NAME	UNIQUENESS	STATUS	INDEX_TYPE	JOIN_INDEX	COLUMNS
1	ISIS2304B061610	SALIDAS_PK	UNIQUE	VALID	NORMAL	NO	ID_BUQUE, ID_MUELLE, FECHA

Buques:

	INDEX_OWNER	INDEX_NAME	UNIQUENESS	STATUS	INDEX_TYPE	JOIN_INDEX	COLUMNS
1	ISIS2304B061610	BUQUES_PK	UNIQUE	VALID	NORMAL	NO	ID

TipoCarga:

	INDEX_OWNER	INDEX_NAME	UNIQUENESS	STATUS	INDEX_TYPE	JOIN_INDEX	COLUMNS
1	ISIS2304B061610	TIPO_CARGA_PK	UNIQUE	VALID	NORMAL	NO	ID_BUQUE, TIPO_CARGA

Carga:

	INDEX_OWNER	INDEX_NAME	UNIQUENESS	STATUS	INDEX_TYPE	JOIN_INDEX	COLUMNS
1	ISIS2304B061610	CARGA_PK	UNIQUE	VALID	NORMAL	NO	ID

Movimiento Marítimo:

	INDEX_OWNER	INDEX_NAME	UNIQUENESS	STATUS	INDEX_TYPE	JOIN_INDEX	COLUMNS
1	ISIS2304B061610	MOVIMIENTO...	UNIQUE	VALID	NORMAL	NO	ID_CARGA, FECHA

Movimiento Terrestre:

	INDEX_OWNER	INDEX_NAME	UNIQUENESS	STATUS	INDEX_TYPE	JOIN_INDEX	COLUMNS
1	ISIS2304B061610	MOVIMIENTO...	UNIQUE	VALID	NORMAL	NO	TIPO, FECHA, ID_CARGA

Los índices son creados automáticamente por Oracle 11g pues se trata de columnas seleccionadas como parte de la primary key de cada tabla. Además de esto, Oracle 11g crea índices para todas las columnas seleccionadas como UNIQUE, empleando siempre índices de tipo NORMAL, que en Oracle 11g corresponden a de tipo B-tree. Estos índices ayudan al desempeño de las sentencias que corresponden a encontrar un cierto valor dado por las PKs, además de cuando se trata de filtrar por valores usando relaciones de mayor o menor (>, <, =>, etc) especialmente cuando se incluyen fechas.

2 Documentación del análisis por requerimiento

2.1 RFC7

2.1.1 Escenario de pruebas

- Sentencia SQL:

```
SELECT arr.*
FROM ARRIBOS arr INNER JOIN (
    SELECT b.ID, b.NOMBRE, b.TIPO, t.TIPO_CARGA
    FROM BUQUES b INNER JOIN TIPOCARGA t ON(b.ID=t.ID_BUQUE)
    WHERE b.NOMBRE='Harry'
        AND b.TIPO = 'RORO'
        AND t.TIPO_CARGA = 'nonummy'
)ta ON (arr.ID_BUQUE=ta.ID)
WHERE FECHA BETWEEN to_date('2015-05-20 4:00:00') AND
to_date('2016-01-01 4:00:00')
ORDER BY(FECHA);
```

- En la tabla buques, la distribución de los datos de prueba es aleatoria, tomando según un rango de datos restringido cuando es necesario. En TIPO, la distribución aleatoria toma valores aleatorios en el rango necesario de (RORO, PORTA_CONTENEDOR, MULTI_PROPOSITO). Las fechas de arribos también están tomadas en un rango aleatorio, por lo que su distribución cuando se filtran las fechas depende únicamente de la amplitud del rango seleccionado, pues entre más amplio es este más tuplas se obtienen en la respuesta.
- Para este análisis se utilizaron los parámetros 'Harry', 'RORO', 'nonummy' y '2015-05-20 4:00:00', '2016-01-01 4:00:00' como el nombre del buque, el tipo del buque, el tipo de carga y el rango de fechas respectivamente. Además el parámetro para ordenar es la fecha.
- **Plan de consulta:**

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		2	5
SORT (ORDER BY)		2	5
HASH JOIN		2	4
Access Predicates			
ARR.ID_BUQUE=B.ID			
NESTED LOOPS		2	4
NESTED LOOPS		2	4
STATISTICS COLLECTOR			
HASH JOIN		1	3
Access Predicates			
B.ID=T.ID_BUQUE			
Filter Predicates			
B.ID=T.ID_BUQUE			
NESTED LOOPS		1	3
STATISTICS COLLECTOR			
BITMAP CONVERSION (TO ROWIDS)		6	2
BITMAP INDEX (RANGE SCAN)	TIPOCARGA_I...		
Access Predicates			
T.TIPO_CARGA='nonumm'			
Filter Predicates			
T.TIPO_CARGA='nonumm'			
TABLE ACCESS (BY INDEX ROWID)	BUQUES	1	1
Filter Predicates			
AND			
B.NOMBRE='Harry'			
B.TIPO='RORO'			
INDEX (UNIQUE SCAN)	BUQUES_PK	1	0
Access Predicates			
B.ID=T.ID_BUQUE			
TABLE ACCESS (BY INDEX ROWID BATCH)	BUQUES	1	1
BITMAP CONVERSION (TO ROWIDS)			
BITMAP AND			
BITMAP CONVERSION (FROM RC			
INDEX (RANGE SCAN)	BUQUES_INDE...	1	1
Access Predicates			
B.NOMBRE='Harry'			
BITMAP INDEX (SINGLE VALUE)	BUQUES_INDE...		
Access Predicates			
B.TIPO='RORO'			
INDEX (RANGE SCAN)	ARRIBOS_PK	1	1
Access Predicates			
AND			
ARR.ID_BUQUE=B.ID			
ARR.FECHA>=TO_DATE(' 2015-05-2			
ARR.FECHA<=TO_DATE(' 2016-01-0			
TABLE ACCESS (BY INDEX ROWID)	ARRIBOS	1	1
TABLE ACCESS (FULL)	ARRIBOS	1	1
Filter Predicates			
AND			
ARR.FECHA>=TO_DATE(' 2015-05-20 04:(
ARR.FECHA<=TO_DATE(' 2016-01-01 04:(

- El tiempo para acceder a los datos, según SQL Developer, oscila entre 0,027 y 0,053 segundos.

2.1.2 Eficiencia

- Algunos escenarios posibles corresponden a utilizar un rango de fechas más limitado para filtrar más datos (por ejemplo, utilizar 2015-05-20 como límite inferior de la fecha resulta en la mitad de los datos respecto a los ejemplos arriba) o utilizar otros en nombre, tipo y tipo carga (por ejemplo, Aaron, RORO y Tortor también resultan en la mitad de número de datos del primer ejemplo).
- Para el escenario base, el procedimiento empezaría por ejecutar el INNER JOIN, mediante un algoritmo de HASH JOIN, sobre las tablas de BUQUES y

TIPO_CARGA sobre el ID del buque. Con este resultado, recorridos utilizando los índices propuestos filtran los predicados por Nombre, Tipo y Tipo_Carga tal que a partir del Join inicial se obtiene un conjunto de tuplas para ser ahora juntado con Arribos. Esto se hace, de nuevo, mediante un Hash Join, y esto se filtra por último con recorridos utilizando los índices de B+ sobre los rangos de fechas, para obtener el resultado final.

- Nuestro plan propuesto es similar al propuesto por Oracle, aunque el plan de Oracle es más específico frente a los filtros usando índices y tiene pasos adicionales sobre los tipos de índices específicos y accesos separados para algunos tipos de índices.

2.2 RFC8

2.2.1 Escenario de pruebas

- Sentencia SQL:

```
SELECT arr.* FROM ARRIBOS arr
INNER JOIN
  (SELECT b.ID, b.NOMBRE, b.TIPO, t.TIPO_CARGA
   FROM BUQUES b INNER JOIN TIPOCARGA t ON(b.ID=t.ID_BUQUE)
   WHERE b.ID NOT IN
     (SELECT b.ID FROM BUQUES b
      INNER JOIN TIPOCARGA t ON(b.ID=t.ID_BUQUE)
      WHERE b.NOMBRE = 'Harry'
      AND b.TIPO = 'RORO'
      AND t.TIPO_CARGA = 'nonummy'
     )
  )ta ON(arr.ID_BUQUE=ta.ID)
WHERE FECHA < to_date('2015-05-20 4:00:00')
OR FECHA > to_date('2016-01-01 4:00:00')
ORDER BY(FECHA);
```

- En la tabla buques, la distribución de los datos de prueba es aleatoria, tomando según un rango de datos restringido cuando es necesario. En TIPO, la distribución aleatoria toma valores aleatorios en el rango necesario de (RORO, PORTA_CONTENEDOR, MULTI_PROPOSITO). Las fechas de arribos también están tomadas en un rango aleatorio, por lo que su distribución cuando se filtran las fechas depende únicamente de la amplitud del rango seleccionado, pues entre más amplio es este más tuplas se obtienen en la respuesta.
- Para este análisis se utilizaron los parámetros 'Harry', 'RORO', 'nonummy' y '2015-05-20 4:00:00', '2016-01-01 4:00:00' como el nombre del buque, el tipo del buque, el tipo de carga y el rango de fechas respectivamente. Además para ordenar se usará la fecha.
- **Plan de consulta:**

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		5571	28
SORT (ORDER BY)		5571	28
HASH JOIN		5571	27
Access Predicates			
T.ID_BUQUE=ARR.ID_BUQUE			
INDEX (FAST FULL SCAN)	TIPO_CARGA_PK	4680	13
HASH JOIN (RIGHT ANTI)		4646	14
Access Predicates			
ARR.ID_BUQUE=ID			
VIEW	VW_NSO_1	6	3
NESTED LOOPS (SEMI)		6	3
BITMAP CONVERSION (TO ROWIDS)		6	2
BITMAP INDEX (RANGE SCAN)	TIPOCARGA_IN...		
Access Predicates			
T.TIPO_CARGA='nonummy'			
Filter Predicates			
T.TIPO_CARGA='nonummy'			
TABLE ACCESS (BY INDEX ROWID)	BUQUES	23	1
Filter Predicates			
AND			
B.NOMBRE='Harry'			
B.TIPO='RORO'			
INDEX (UNIQUE SCAN)	BUQUES_PK	1	0
Access Predicates			
B.ID=T.ID_BUQUE			
TABLE ACCESS (FULL)	ARRIBOS	4653	11
Filter Predicates			
OR			
ARR.FECHA<TO_DATE(' 2015-05-20 04:00:00', 'YYYY-MM-DD HH24:MI:SS')			
ARR.FECHA>TO_DATE(' 2016-01-01 04:00:00', 'YYYY-MM-DD HH24:MI:SS')			

- El tiempo de ejecución de este caso oscila entre 0,027 y 0,4 segundos, según SQL Developer.

2.2.2 Eficiencia

- Algunos escenarios posibles corresponden a utilizar un rango de fechas más limitado para filtrar más datos (por ejemplo, utilizar 2015-05-20 como límite inferior de la fecha resulta en la mitad de los datos respecto a los ejemplos arriba) o utilizar otros en nombre, tipo y tipo carga (por ejemplo, Aaron, RORO y Tortor también resultan en la mitad de número de datos del primer ejemplo).
- Para el escenario base, el procedimiento empezaría por ejecutar el INNER JOIN, mediante un algoritmo de HASH JOIN, sobre las tablas de BUQUES y TIPO_CARGA sobre el ID del buque. Con este resultado, ya que es necesario filtrar y sacar los elementos que correspondan a los dados en los parámetros, se filtra con índices de los predicados. Este resultado finalmente se utiliza para juntar con la tabla arribos. Posteriormente, se filtra utilizando los predicados de fechas con un rango, para obtener el resultado final.
- El plan planteado por Oracle es similar al planteado por nosotros en la idea general, aunque Oracle ejecuta la instrucción NOT IN utilizando un HASH JOIN RIGHT ANTI, mientras que nuestro plan utiliza un acceso y filtrado simple usando los

índices. El plan de Oracle resulta más eficiente pues el filtrado usando índices es menos eficiente cuando se trata de conjuntos de datos a intersecar o a restar, como en este caso que se usa una instrucción NOT IN.

2.3 RFC9

2.3.1 Escenario de pruebas

```
- Sentencia SQL:
SELECT c.ORIGEN, c.DESTINO, c.ID as ID_CARGA, c.TIPO, m.FECHA as
FECHA_MOVIMIENTO FROM
CARGA c INNER JOIN
(
    SELECT ID_CARGA, FECHA FROM MOVIMIENTOMARITIMO
    UNION
    SELECT ID_CARGA, FECHA FROM MOVIMIENTOTERRESTRE
) m on(c.ID=m.ID_CARGA)
WHERE c.ID_EXPORTADOR IN
(
    SELECT ID_EXPORTADOR FROM
    CARGA WHERE
    VALOR>7500
)
AND c.TIPO = 'Ropa';
```

- La distribución de los datos de Carga correspondió a un juego de datos aleatorio con algunas restricciones en ciertos casos para hacer las tuplas más significativas a la hora de hacer las sentencias. Algunos valores, como el número y el volumen, se generaron aleatoriamente con ciertos rangos y aproximaciones decimales. Los IDs de personas se generaron aleatoriamente siguiendo las restricciones de FK de las tablas. En los Tipos, se siguió una distribución aleatoria entre un rango de valores reducidos, para ser filtrado según los datos correspondientes en el requerimiento.

Para este caso, el tipo y el valor de las cargas a discriminar por exportador son los parámetros que afectan la distribución de los resultados. Para Tipo, ya que este se hizo aleatoriamente sobre un rango de valores en la configuración de los datos de entrada, estos están distribuidos uniformemente en los resultados. Para valor, entre mayor es el rango de valores, menores son los resultados y el tiempo es menor, pues se filtran más valores en una sola operación.

- Para este análisis se utilizaron los parámetros 'Harry', 'RORO', 'nonummy' y '2015-05-20 4:00:00', '2016-01-01 4:00:00' como el nombre del buque, el tipo del buque, el tipo de carga y el rango de fechas respectivamente.
- **Plan de consulta:**

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		3502	97
HASH JOIN		3502	97
Access Predicates			
C.ID=M.ID_CARGA			
HASH JOIN (RIGHT SEMI)		2625	75
Access Predicates			
C.ID_EXPORTADOR=ID_EXPORTADOR			
VIEW	index\$_join\$_006	2175	45
Filter Predicates			
VALOR>7500			
HASH JOIN			
Access Predicates			
ROWID=ROWID			
INDEX (RANGE SCAN)	CARGA_INDEX_...	2175	21
Access Predicates			
VALOR>7500			
INDEX (FAST FULL SCAN)	CARGA_INDEX_I...	2175	30
TABLE ACCESS (BY INDEX ROWID BATCHED)	CARGA	5566	30
BITMAP CONVERSION (TO ROWIDS)			
BITMAP INDEX (SINGLE VALUE)	CARGA_INDEX_...		
Access Predicates			
C.TIPO='Ropa'			
VIEW		14621	22
SORT (UNIQUE)		14621	22
UNION-ALL			
INDEX (FAST FULL SCAN)	MOVIMIENTOMA...	5914	9
TABLE ACCESS (FULL)	MOVIMIENTOTE...	8707	11

- Para este caso, los tiempos oscilan entre 0.04 y 0.113 segundos, según SQL Developer.

2.3.2 Eficiencia

- Otros escenarios que permiten analizar los datos de respuesta corresponden a cambiar los parámetros de tipo y valor. En el cambio de valor especialmente, los rangos menores de datos ralentizan el plan de ejecución de consulta significativamente, generando un mayor costo de ejecución entre una mayor proporción de los datos se encuentra fuera del rango establecido por este parámetro. Por ejemplo, la siguiente imagen muestra la diferencia entre un valor escogido de 3000 y el del base de 7500:

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		7426	175
HASH JOIN (RIGHT SEMI)		7426	175
Access Predicates			
C.ID_EXPORTADOR=ID_EXPORTADOR			
VIEW	index\$_join\$_006	10841	123
Filter Predicates			
VALOR>100			
HASH JOIN			
Access Predicates			
ROWID=ROWID			
INDEX (RANGE SCAN)	CARGA_INDEX_VALOR	10841	99
Access Predicates			
VALOR>100			
INDEX (FAST FULL SCAN)	CARGA_INDEX_IDEXP	10841	30
HASH JOIN		7426	52
Access Predicates			
C.ID=M.ID_CARGA			
TABLE ACCESS (BY INDEX ROWID BATCHED)	CARGA	5566	30
BITMAP CONVERSION (TO ROWIDS)			
BITMAP INDEX (SINGLE VALUE)	CARGA_INDEX_TIPO		
Access Predicates			
C.TIPO='Ropa'			
VIEW		14621	22
SORT (UNIQUE)		14621	22
UNION-ALL			
INDEX (FAST FULL SCAN)	MOVIMIENTOMARITL...	5914	9
TABLE ACCESS (FULL)	MOVIMIENTOTERRES...	8707	11

- En el escenario base, el plan de ejecución propuesto empieza por la unión de las tablas Movimiento_Terrestre y Movimiento_Maritimo, seguido de un Hash Join con la tabla de Carga. Tras esto, se hace otro Hash Join con la tabla previamente filtrada por el índice B+ de valor para obtener un conjunto de exportadores. Este resultado, finalmente, se filtra mediante el índice de TIPO para obtener sólo las tuplas que correspondan a dicho parámetro.
- El plan de Oracle realiza un HASH JOIN RIGHT SEMI para los valores correspondientes a buscar las tuplas con exportador que estén entre los que tienen cargas con un valor mayor al dado. Los dos planes de ejecución son similares en la idea general.

2.4 RFC10

2.4.1 Escenario de pruebas

- Sentencia SQL:

```

SELECT ID, TIPO as TIPO_CARGA,ORIGEN as ORIGEN_CARGA, DESTINO as
DESTINO_CARGA,FECHA, TIPO_MOV, ID_ALMACENAMIENTO  FROM
CARGA INNER JOIN(
  SELECT ID_CARGA, FECHA, TIPO||' respecto al buque' as TIPO_MOV,
ID_ALMACENAMIENTO
  FROM MOVIMIENTOMARITIMO
  UNION
  SELECT ID_CARGA, FECHA, TIPO||' respecto al almacen' as TIPO_MOV,
ID_ALMACEN
  FROM MOVIMIENTOTERRESTRE
) ON (ID=ID_CARGA)

```

WHERE ID_ALMACENAMIENTO=9 OR ID_ALMACENAMIENTO=1;

- La distribución de los datos de Carga correspondió a un juego de datos aleatorio con algunas restricciones en ciertos casos para hacer las tuplas más significativas a la hora de hacer las sentencias. Algunos valores, como el número y el volumen, se generaron aleatoriamente con ciertos rangos y aproximaciones decimales. Los IDs de personas se generaron aleatoriamente siguiendo las restricciones de FK de las tablas. En los Tipos, se siguió una distribución aleatoria entre un rango de valores reducidos, para ser filtrado según los datos correspondientes en el requerimiento.
- Para este análisis se utilizaron los valores de parámetro del almacenamiento con ID 1 y el almacenamiento con ID 9.
- **Plan de consulta:**

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		31	53
HASH JOIN		31	53
Access Predicates ID=ID_CARGA			
NESTED LOOPS		31	53
NESTED LOOPS		31	53
STATISTICS COLLECTOR			
VIEW		31	22
SORT (UNIQUE)		31	22
UNION-ALL			
TABLE ACCESS (FULL)	MOVIMIENTO...	14	9
Filter Predicates			
OR			
ID_ALMACENAMIENTO=1			
ID_ALMACENAMIENTO=9			
TABLE ACCESS (FULL)	MOVIMIENTOT...	17	11
Filter Predicates			
OR			
ID_ALMACEN=1			
ID_ALMACEN=9			
INDEX (UNIQUE SCAN)	CARGA_PK	1	0
Access Predicates ID=ID_CARGA			
TABLE ACCESS (BY INDEX ROWID)	CARGA	1	1
TABLE ACCESS (FULL)	CARGA	1	1

- El tiempo varía entre 0.014 y 0.126 segundos según SQL Developer.

2.4.2 Eficiencia

- Cambiar los almacenamientos involucrados con sus IDs en los parámetros resulta en escenarios distintos pero que coinciden aproximadamente en tamaño a los que resultan del escenario base descrito arriba.

- En el escenario base descrito arriba, nuestro plan de ejecución propuesto es el siguiente: En primer lugar, se hace una unión de las tablas correspondientes a los movimientos. Seguidamente, un HASH JOIN con Carga junta los valores de estas tablas mediante el ID y, finalmente, se filtran los datos por ID_almacenamiento, utilizando los índices definidos.
- El plan de Oracle difiere en que éste realiza el filtrado por predicados en primer lugar, al contrario de una vez se han unido y juntado las tablas con HASH JOIN, como lo hace nuestro plan. En este caso, el plan de Oracle es más eficiente pues puede filtrar una menor cantidad de datos si lo hace antes de unir las tablas, en lugar de después, pues estas sólo ocurren en estas tablas 'interiores'.

3 Carga de Tablas

Las tablas se cargaron usando el paquete de Oracle PL/SQL dbms_random para generar valores aleatorios en el conjunto de datos de las tablas, con la siguiente sentencia:

```
insert into buques
select
  rownum,
  initcap(dbms_random.string('l',dbms_random.value(2,10))),
  dbms_random.string('l',dbms_random.value(2,20)),
  initcap(dbms_random.string('l',dbms_random.value(2,10))),
  dbms_random.string('l',dbms_random.value(2,20)),
  (select string FROM
    (select string
     from (select string from strings order by dbms_random.value) s
     where rownum = 1
    )),

  round(dbms_random.value(1,1000),0),
  'PUERTO_' || initcap(dbms_random.string('l',dbms_random.value(2,9)))
from
  (select level from dual connect by level <= 1000000);
```

Donde strings corresponde a una tabla con los valores posibles correspondientes al atributo de la tabla buques. Esto nos permitió cargar las tablas muy rápidamente y con valores apropiados.

```
insert into carga
select
  rownum,
  initcap(dbms_random.string('l',dbms_random.value(2,10))),
  round(dbms_random.value(1,4044),0),
  round(dbms_random.value(1,100),0),
  round(dbms_random.value(1,500),1),
  round(dbms_random.value(1,4044),0),
```

```

'PUERTO_' || initcap(dbms_random.string('l',dbms_random.value(2,9))),
0,
1,
'PUERTO_' || initcap(dbms_random.string('l',dbms_random.value(2,9))),
round(dbms_random.value(1,500),1),
round(dbms_random.value(1,10000),2)
from
(select level from dual connect by level <= 1000);

```

También utilizamos la herramienta online *mockaroo*, servicio que nos permitió crear datos con valores más personalizados para así crear datos coherentes con el contexto de la aplicación. Esta herramienta también nos permitió crear distribuciones más acertadas pues permitía usar funciones para la creación de los datos. El único limitante de esta aplicación es que permitía crear tan solo 1000 sentencias de inserción, para hacer más tocaba repetir la descarga varias veces. Entonces para cada tabla que se utiliza en los requerimientos se crearon 10,000 datos con *mockaroo*, esto con el propósito de que hubiesen más datos apropiados al contexto (por ejemplo, en la tabla TipoCarga había fecha_ingreso y fecha_retiro, atributos que *mockaroo* permite restringir para que el retiro no fuese antes del ingreso).

4 Comparación de ejecución de consultas

Como observamos en las pruebas de POSTMAN en comparación a las pruebas realizadas en SQL Developer, cuando se traen datos a memoria principal las operaciones de los datos resultan mucho menos eficientes, pues las estructuras de datos que utiliza el Manejador de Base de Datos son mucho más eficientes que las que puede realizar Java sobre volúmenes de datos grandes. Además, con necesidades como los Joins y selecciones, iterar sobre los datos y manipularlos usando un flujo de control requiere más recursos y consume más tiempo, pues tiene que lidiar con la sobrecarga tanto temporal como en espacio de las clases, las instancias de objetos, y otros elementos que utiliza Java para abstraer y utilizar los datos con facilidad.

Además, cabe resaltar que ciertas operaciones de Oracle se ejecutan de una manera que no es posible (o por lo menos no es fácil de implementar) con instrucciones de control. Esto se observa más cuando se utilizan índices. Por ejemplo, hacer Join mediante Hash se hace rápido tanto en sentencias como en velocidad de ejecución, además es eficiente, por otro lado, intentar implementar algo similar con instrucciones de control tomaría más tiempo y seguramente el computador utilizará más recursos.