**Taller 4 - Optimización de Consultas de una Aplicación Transaccional**

Jairo Emilio Bautista Mora, Nelson Andrés Sánchez Otálora

Caso de Estudio: ProdAndes

Universidad de los Andes, Bogotá, Colombia

{je.bautista10, na.sanchez162}@uniandes.edu.co

Fecha de presentación: Mayo 3 de 2013

Tabla de contenido

[1. Análisis 1](#_Toc418452686)

[2. Diseño de la aplicación 1](#_Toc418452687)

[1. Análisis de impacto de Introducción de nuevos requerimientos 1](#_Toc418452688)

[2.2. Diseño Físico: Selección de Índices 2](#_Toc418452689)

[3. Construcción de la aplicación y análisis de resultados 7](#_Toc418452690)

[3.4 Análisis del Proceso de Optimización y de Ejecución de Consultas 8](#_Toc418452691)

# Análisis

De acuerdo al modelo relacional en la iteración anterior, y los nuevos requerimientos funcionales de consulta solicitados, no se hizo ningún cambio en el modelo relacional del mundo. No se agregaron ningún atributo ni ninguna nueva relación.

# Diseño de la aplicación

## 2.1 Análisis de impacto de Introducción de nuevos requerimientos

Para cada uno de los nuevos requerimientos funcionales de consulta solicitados se hizo un análisis de impacto correspondiente:

* RFC8 - Consultar ejecución de etapas de producción 1:

Dado que se requiere retornar las etapas de producción dado un rango de tiempo, y por criterios de dicha etapa (material, tipo de material, pedido, cantidades,…). Con el modelo actual, se puede alcanzar este requerimiento ya que solo requiere de hacer los JOINs necesarios para filtrar la información solicitada.

* RFC9 - Consultar ejecución de etapas de producción 2:

De igual forma que el requerimiento anterior; con el modelo actual, basta con realizar el complemento de la consulta anterior para obtener el resultado esperado.

* RFC10 - Consultar pedidos 2:

De acuerdo al modelo actual, es posible poder buscar los materiales requeridos por medio de condiciones WHERE dentro de los atributos de la respectiva tabla. Por tanto, no es necesario cambiar el modelo para cumplir este requerimiento funcional.

* RFC11 - Consultar material 2:

De acuerdo al modelo actual, para cumplir con las consultas de este requerimiento, basta con realizar los respectivos JOINS entre materiales, etapas de producción, y pedidos; de tal forma que se pueda mostrar la información completa del material según las especificaciones solicitadas.

* RNF5 - Eficiencia en las consultas:

Si bien este requerimiento no exige un cambio al nivel del modelo del mundo, si exige unas variaciones en la forma de implementar la base de datos de la aplicación. En particular, se requieren agregar índices a las tablas de la base de datos; con el fin de volver más eficientes las consultas realizadas a la aplicación.

* RNF6 - Eficiencia en la actualización:

De forma similar al anterior, se necesita validar el tipo de índices en las tablas de las base de datos, con el fin de verificar la eficiencia en las actualizaciones, y general un equilibrio aceptable de esta con respecto a la eficiencia de realizar consultas.

* RNF7 - Esquema físico de la base de datos:

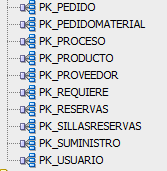
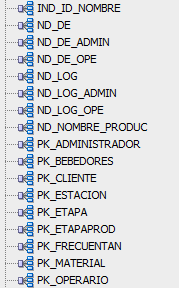
En concordancia con los dos requerimientos anteriormente analizados, se requiere verificar el manejo de índices, de tal forma que se pueda mantener una entre las eficiencias de consulta y actualización de acuerdo a los exigidos en el enunciado.

Adicionalmente, se requiere poblar la base de datos con un número significativo de datos con el fin de poder la validar el correcto funcionamiento de los requerimientos exigidos para la presente iteración.

## 2.2. Diseño Físico: Selección de Índices

Adaptando los índices que genera la aplicación de Oracle por default, y agregando incides adicionales propios de la aplicación, es obtienen en total los siguientes índices:

Los creados inicialmente por Oracle:



Y los generados por el equipo de desarrollo:

* EtapaProducción (FK EnEsperaDe (EstaciónProduccion.Codigo))
* Producto (Cantidad, Costos)
* Material (UltimoAbastecimiento, Cantidad)
* EtapaProduccion ( FechaInicio, FechaFin, Nombre, CantidadProducto)
* Suministros (tiempoEntrega, maximaCantidad)
* Pedido (fechaPedido)

De los cuales, se pueden resaltar los siguientes aspectos:

### 2.2.1. Índices en llaves primarias e indices de atributos no duplicables (por defecto Oracle):

En primer lugar, teniendo en cuenta que los atributos de las llaves primarias de una tabla identifican unívocamente cada tuplas; estos los vuelve candidatos ideales para ser índices. Adicionalmente, como los índices es el principal criterio para la realización de JOINs de la aplicación, utilizarlos aumentará considerablemente la eficiencia de las consultas de la aplicación. Vemos que estos índices son creados por default de Oracle

Adicionalmente, se puede observar como Oracle crea índices en los atributos no duplicables, en razón a que estos pueden identificar unívocamente una tupla de una tabla. Por tanto, son candidatos ideales a índices en la aplicación.

### 2.2.2. Índices no únicos en llaves foráneas.

Si bien ya se han establecido índices a las PK de las tablas, también se requiere establecer en consultas particulares índices en las llaves foráneas de algunas tablas, esto debido a la metodología en las cuales se implementan los requerimientos funcionales.

Específicamente, en el requerimiento funcional RF17; se requiere tener un índice en la tabla EtapaProducción para el atributo de llave foránea EnEsperaDe. Esto debido a que este atributo es frecuentemente usado para recuperar las tablas necesarias para realizar el balanceo necesario en las etapas de producción; a la hora de cambiar el estado de la estación de producción correspondiente.

En razón a que el número posible de variables de este atributo depende directamente del número posible de PK de EstaciónDeProducción; la variabilidad de estos datos será alta, y por tanto se decide usar un B-Tree Index en este caso.

* EtapaProducción (FK EnEsperaDe (EstaciónProduccion.Codigo))

### 2.2.3 Índices Compuestas

Adicional a los JOINs que se requieren para los requerimientos funcionales, también un número significativo de consultas requieren realizar filtros de las tuplas posibles; estos filtros varían desde ordenamientos por fecha, cantidad, costos, entre otros. Por tanto, se propone manejar índices compuestos en las tablas que requieran estas consultas adicionales a los índices únicos de llaves primarias. Estas llaves compuestas reunirán todas las columnas que sirvan de filtro en las condiciones WHERE de consulta, y tendrán como función agilizar el filtrado de las tuplas de acuerdo a los parámetros del usuario.

Para el caso de los atributos presentados a continuación, se puede observar cómo los datos mostrados no cambian frecuentemente en el tiempo, salvo por cantidad. Adicionalmente, los datos presentan una alta variabilidad en su información. Por tanto, se decide usar B-Tree Index en este caso.

Específicamente, las llaves compuestas usadas serán:

* Producto (Cantidad, Costos)
* Material (UltimoAbastecimiento, Cantidad)
* EtapaProduccion ( FechaInicio, FechaFin, Nombre, CantidadProducto)
* Suministros (tiempoEntrega, maximaCantidad)

### 2.2.4 Índices no únicos

Finalmente, en las consultas que implican el uso de la tabla de pedidos, el usuario tiene la opción de filtrar las consultas por medio de la fecha de pedido, por tanto, se decide agregarle un índice a esta última, con el fin de optimizar el proceso de filtrado.

De forma similar, como las fechas de pedido es una información altamente variable en la aplicación, se decide usar B-Tree Index en este caso.

* Pedido (fechaPedido)

## 2.3. Documentación de los Escenarios de Prueba

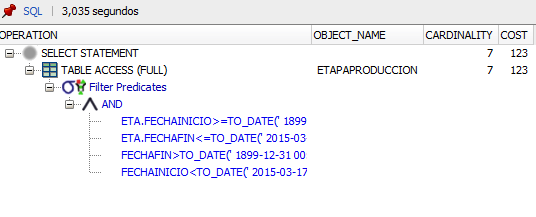
* RFC8:

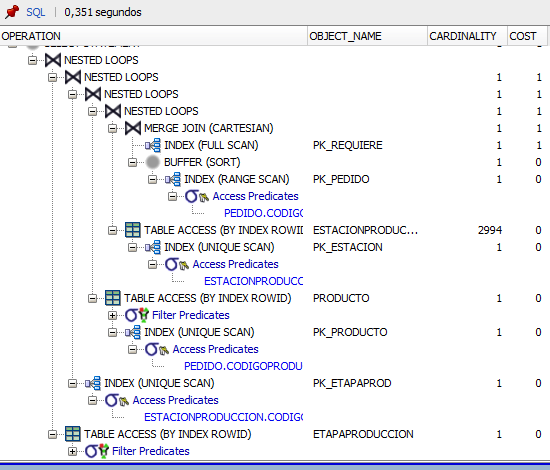
SELECT eta.\* FROM EtapaProduccion eta WHERE '31-12-1899'<=eta.FechaInicio AND '17-03-2015'>=eta.FechaFin;

Esta consulta nos permite obtener las etapas de producción que cumplen con los parámetros datos por el usuario, en este caso correspondiendo a 31/12/1899 y a 17/03/2015. Los datos en la base de datos están entre el 2012 y el 2012, con respecto a fechas, y se agregaron uniformemente en la base de datos. Estos parámetros generan que la consulta tenga una selectividad baja, pero según los parámetros dados por el usuario, la selectividad puede aumentar, dejando un rango muy pequeño, creciendo de manera lineal.

SELECT eta.\* FROM PEDIDO, Producto, EtapaProduccion eta , EstacionProduccion, Requiere, Material WHERE '31-12-1899'<=eta.FechaInicio AND '17-03-2015'>=eta.FechaFin AND EstacionProduccion.CodigoEtapa=eta.Codigo AND EstacionProduccion.Codigo=Requiere.CodigoEstacion AND Material.Codigo=Requiere.CodigoMaterial AND Material.Tipo='Materia Prima' AND Material.Codigo=1 AND PEDIDO.CodigoProducto=Producto.Codigo AND Producto.Codigo= eta.CODIGOPRODUCTO AND PEDIDO.Codigo=1 AND Producto.Cantidad>=0 AND Producto.Cantidad<=1000000;

Esta consulta es la misma anterior, pero enriquecida con los demás filtro para reducir la cantidad de resultados obtenidos. Aunque esto indica que se usaran algunos índices creados en la base de datos. Reduciendo la cantidad de cantidades de producto en inventario, indicando cual producto y cual material esta involucrado dentro de esa etapa la cantidad de resultados permitiría aumentar la selectividad al menos en un 50%.





* RFC9:

select eta2.\* from (select \* from ETAPAPRODUCCION) eta2 MINUS (SELECT eta.\* FROM EtapaProduccion eta WHERE '31-12-1899'<=eta.FechaInicio AND '17-03-2015'>=eta.FechaFin);

select eta2.\* from (select \* from ETAPAPRODUCCION) eta2 MINUS (SELECT eta.\* FROM PEDIDO, Producto, EtapaProduccion eta , EstacionProduccion, Requiere, Material WHERE '31-12-1899'<=eta.FechaInicio AND '17-03-2015'>=eta.FechaFin AND EstacionProduccion.CodigoEtapa=eta.Codigo AND EstacionProduccion.Codigo=Requiere.CodigoEstacion AND Material.Codigo=Requiere.CodigoMaterial AND Material.Tipo='Materia Prima' AND Material.Codigo=1 AND PEDIDO.CodigoProducto=Producto.Codigo AND Producto.Codigo= eta.CODIGOPRODUCTO AND PEDIDO.Codigo=1 AND Producto.Cantidad>=0 AND Producto.Cantidad<=1000000);

Al contrario de la consulta anterior, entre más filtros más tiempo va a tardar y más registros van a conseguir. Los valores de los parámetros ingresados por el usuario aumentan la cantidad de registros. Entre más exactos sean los rangos o específicos los valores únicos, más grande en el conjunto de resultado, creciendo de manera lineal. Si no se usan parámetros sino solo los de las fechas, serán solo el rango de fechas dado por el usuario los que indicaran cuales registros no mostrar.

* RFC10:

SELECT ped.\* FROM PEDIDO ped, Producto, EtapaProduccion, EstacionProduccion, Requiere, Material WHERE ped.CodigoProducto=Producto.Codigo AND Producto.Codigo=EtapaProduccion.CodigoProducto AND EstacionProduccion.CodigoEtapa=EtapaProduccion.Codigo AND EstacionProduccion.Codigo=Requiere.CodigoEstacion AND Material.Codigo=Requiere.CodigoMaterial AND Material.Tipo='&paramTipo' AND Producto.Costo>&paramCosto;

En este requerimiento solo hay un filtro, el cual es el costo de un producto, dando una selectividad muy alta según el costo recibido. Asumiendo que los pedidos por parte de los clientes son del mismo orden de magnitud en costo, puede que realmente la selectividad sea muy baja. Según el parámetro recibido el conjunto de respuestas puede crecer o disminuir linealmente. Dado que los datos generados son aleatorios con la misma probabilidad de aparecer, seleccionar un material dado no cambiara mucho el conjunto de resultados.

* RFC11:

SELECT ped.\* FROM PEDIDO ped, Producto, EtapaProduccion, EstacionProduccion, Requiere WHERE ped.CodigoProducto=Producto.Codigo AND Producto.Codigo=EtapaProduccion.CodigoProducto AND EstacionProduccion.CodigoEtapa=EtapaProduccion.Codigo AND EstacionProduccion.Codigo=Requiere.CodigoEstacion AND Requiere.CodigoMaterial= &paramMaterial;

En este requerimiento solo hay un filtro, el cual es el codigo del material, dejando una selectividad muy alta dentro de material, pero dado que ese material puede estar involucrado en varios pedidos, por dado que los datos generados son aleatorios con la misma probabilidad de aparecer, seleccionar un material dado no cambiara mucho el conjunto de resultados.

## 2.4 Análisis de Eficiencia:

Teniendo en cuenta los escenarios planteados en el numeral anterior, se puede plantear el siguiente análisis con respecto al plan de ejecución propuesto en contraste con el plan de ejecución planteado por Oracle en su modo Default.

* RFC8

En este caso, al plantear índices compuestos en el ordenamiento de las columnas que será usadas como filtros (FechaInicio, FechaFin, Cantidad, entre otros); se puede realizar un filtrado más rápido de la información de cuerdo a los rangos usados por parámetro. En contraste, el no haber usado este tipo de índices; hubiera requerido que la sentencia buscara las tuplas requeridas en la tabla de forma dispersa.

Por tanto, y a pesar del alto volumen de los datos presentes en la base de datos, se es posible reducir la latencia de la ejecución del método.

* RFC9

De forma similar al requerimiento anterior, al ser el resultado buscado el complemento de las tuplas resultantes del requerimiento anterior; se puede concluir que las tuplas usadas en el RFC8 nuevamente ayudarán a aumentar el desempeño en la ejecución de RFC9.

* RFC10

Para este requerimiento, al general índices en los costos de productos (usado para calcular el valor de los materiales dentro de la aplicación), se pudo aumentar la eficiencia de las consultas con respecto al caso en el que se usara únicamente los índices de Oracle. Por tanto, el uso de índices en las columnas usadas como filtros, facilitan la consulta y mejora el desempeño del requerimiento.

* RFC11

Igualmente, y adicional a los índices en llaves primarias para el manejo de los JOINs requeridos; el usar Índices de forma Ascendente en las fechas de Pedido facilito el filtrado de la información que se requería hacer en el método; por lo que se consigue aumentar la eficiencia en la consulta de la información deseada.

# 3. Construcción de la aplicación y análisis de resultados

## 3.1 Ajustes:

No se realizaron ajustes.

## 3.2 Diseño de datos para la prueba:

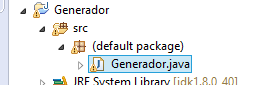
Para las pruebas de eficiencia es necesario cargar una gran cantidad de datos en las tablas utilizadas por la aplicación. Se planeó que hacer la carga de 100000 datos por tabla en 13 de las 17 tablas de la aplicación. Para la verificación de eficiencia se busca traer la mayor cantidad de datos, en menos de 0.8 segundos, sin colapsa la memoria de la maquina (no mostrar todos los datos, sino fragmentos de ellos).

### 3.2.1 Diseño de datos

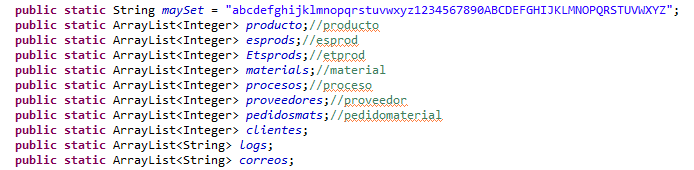
El diseño de los datos de prueba requería que se mantuvieran correctamente las restricciones de llaves foráneas. Los datos que involucraban números enteros fueron generados aleatoriamente con rangos del tamaño y orden para representar los datos de manera congruente. Los datos de texto fueron generados aleatoriamente sin importar su contenido (a menos que tuvieran restricción de valor).

### 3.2.2 Proceso de generación de datos:

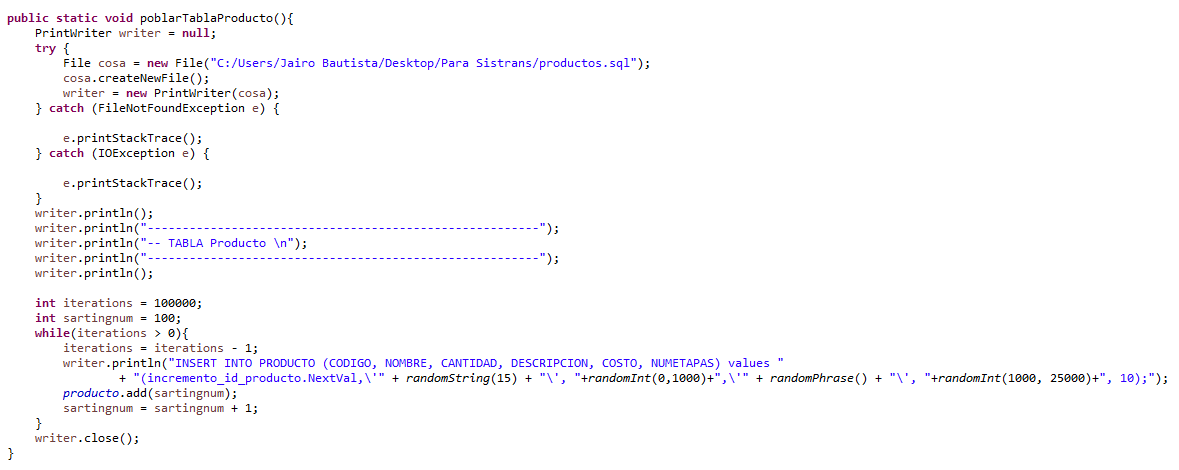
Para la generación de datos se hizo un programa simple en Java, el cual generaba aleatoriamente los tipos de datos necesarios, y los escribía en archivos .SQL, uno por cada tabla que se quería poblar.



En dicho programa se guardaban los códigos y otras llaves principales necesarias entre las tablas para ser usadas como llaves foráneas.



El programa ejecutaba diferentes métodos, uno para cada tabla a poblar, el cual se encargaba de crear un archivo SQL, un printwriter, con eso realizaba 100000 iteraciones imprimiendo una cadena de texto la cual contenía una instrucción de insert respectiva para la tabla.



Al final del programa se habrán generado 13 archivos .sql, los cuales abrimos y ejecutamos como script desde SQLDeveloper. Dado que como los dato se generaron y relacionaron aleatoriamente, no todos los registros pudieron ser creados, pero se asegura que el orden de la cantidad de datos es lo suficiente para que no entren en memoria principal.

## 3.4 Análisis del Proceso de Optimización y de Ejecución de Consultas

En el caso que nuestra aplicación cargara todos los datos de las consultas a memoria principal del servidor de aplicaciones; seria nuestro deber organizarla y trabajar con ella para poder reducir el conjunto de resultados a las correctos para la pregunta dada.

Sin embargo, sería difícil poder cargar toda la información de una consulta, pues por lo general el servidor de aplicaciones tiene menos memoria principal que el servidor en el que se encuentra la base de datos. A demás suponiendo que es una aplicación altamente concurrente, los diferentes usuarios realizaran sus consultas y reducirán aún más el espacio disponible para las consultas de otros usuarios. Si lográramos cargar toda la información en la memoria principal del servidor, sería bastante caro pagar por el espacio extra.

El siguiente problema es que para lograr ordenar toda la información para cada consulta realizada por un usuario, tocaría usar una estructura de datos extra para poder facilitar el proceso (lo cual consumiría espacio extra) e implementar algoritmos eficientes de ordenamiento (que entre más eficiente sea, más complicado de implementar es). Igualmente realizar estos ordenamientos en la aplicación aumentaría bastante la complejidad temporal de ejecución, lo cual bajaría el desempeño y eficiencia del sistema.

Finalmente, usando instrucciones de control para reducir el conjunto de resultados de la aplicación, se pueden presentar demoras por complejidad temporal, dado que tocaría recorrer todo el conjunto de datos ordenado. Tal vez se puedan implementar algoritmos de recorridos para poder lograr un recorrido más eficiente, pero con un gran volumen de datos igual significa tiempo perdido.

Solo para proyectar un conjunto de datos es necesario recorrer todos los demás para sacar los valores proyectados de los registros que cumplen condicionalmente (if...) con la proyección. Una selección sufre lo mismo.

Para un join es necesario ir comparando dos conjuntos mientras se van recorriendo, involucrando un while por cada conjunto, y varios if para verificar la condición de JOIN.

El sistema manejador de bases de datos posee el espacio necesario para poder realizar el ordenamiento involucrando memoria principal y memoria secundaria, además de que ya tiene implementados los algoritmos de ordenamiento y recorrido.

Por tanto, se concluye que es mejor para el desarrollo de la aplicación, delegar estas tareas a la base de datos; la cual está diseñada precisamente para afrontar este tipo de requerimiento. Y de esta forma, el equipo de desarrollo pueda preocuparse por los requerimientos que son propios del servicio que se desea ofrecer al usuario.