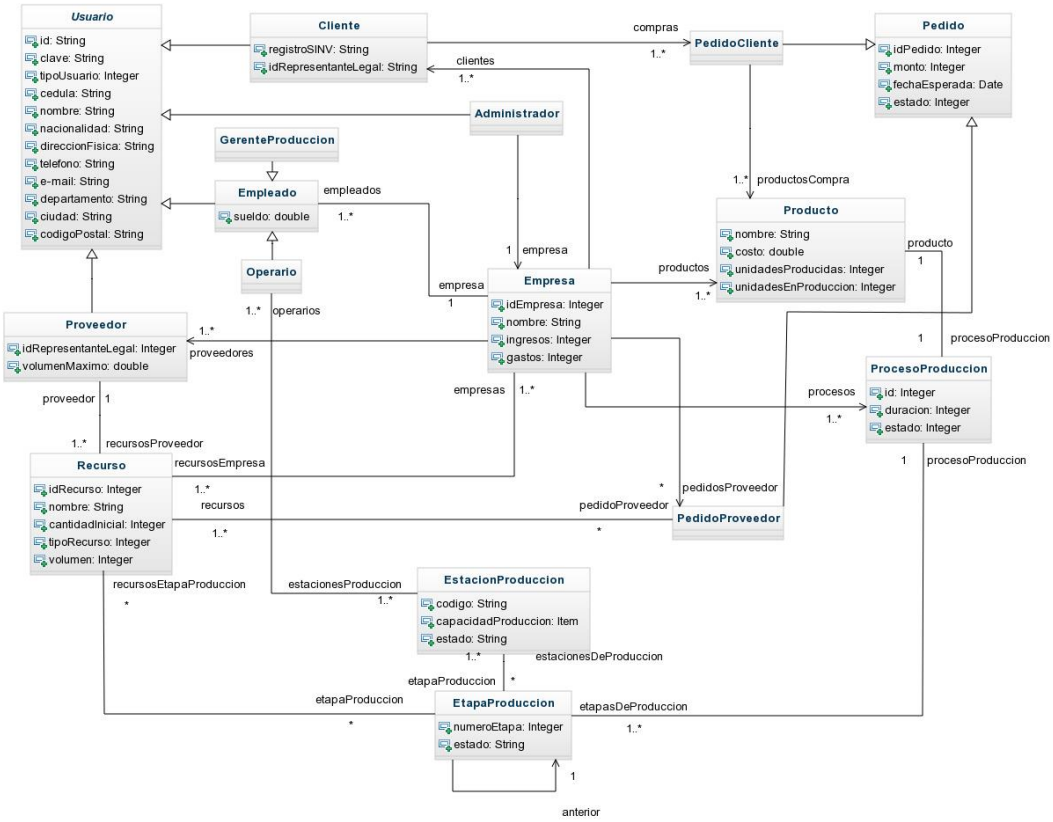
ITERACIÓN 4 – MODELAJE DE UNA APLICACIÓN TRANSACCIONAL
Grupo 3
Carlos Felipe Agudelo Ospina 201328150
Sergio Yodeb Velásquez Yepes 201315851

1. Análisis

Con respecto a la entrega pasada no se realizó ningún cambio al modelo conceptual.



Con respecto al modelo relacional se remplazó la tabla operan con la tabla ejecutaron. Para que se pudiera tener la relación de ejecución de etapas con respecto a una estación de producción, porque en un principio no se había analizado bien esta relación.

Operan

idOperario	idEtapaProduccion	fechaEjecucion
PK1	PK2, FK(EtapasProduccion.idEtapa)	NN
duracion	idPedido	
NN	PK3(Pedido.idPedido)	

Ejecutaron

idEtacionProduccion	idEtapaProduccion	idPedido
PK1, FK(EstacionesProduccion.idEstacionProduccion)	PK2, FK(EtapasProduccion.idEtapa)	PK3,FK(Pedido.idPedido)
TiempoEjecucion	idOperario	fechaEjecucion
NN	PK4, FK(Operarios.idOperario)	NN

Por esta misma razón se agregó la columna IDPEDIDO a la tabla de ejecutan para que hubiera una relación de cada ejecución con un pedido específico.

Ejecutan

IdEstacionProduccion	IdEtapaProduccion	IdPedido
PK1, FK(EstacionesProduccion.idEs)	PK2, FK(EtapasProduccion.idEtapa)	PK3, FK(Pedidos.idPedido)

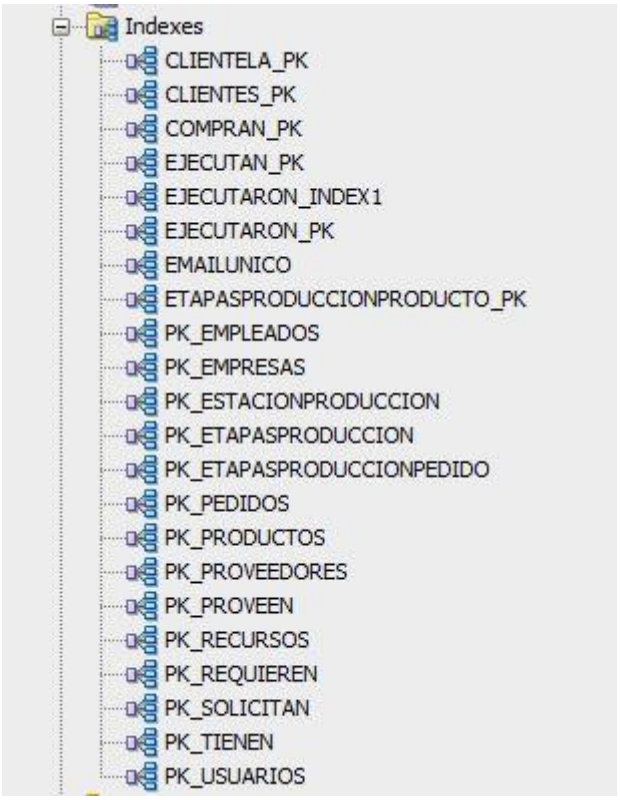
2. Diseño físico

2.1. Documentación del diseño físico

Los índices que manejamos son principalmente los primary key de todas las tablas que son generados por Oracle al nosotros asignarles el valor de primary key. Debido a la diversidad de las consultas crear índices para hacer más eficiente cada una de estas terminaría siendo crear índices para todas las columnas de la base de datos. Por lo cual nosotros miramos que columnas eran las más usadas en todos los requerimientos en general, encontramos que las que más se usaban en nuestro modelo relacional eran la FECHAPEDIDO, la FECHALLEGADA de la tabla Pedidos, el COSTO de la tabla recursos y de la tabla productos, . Puesto que la mayoría de requerimientos tienen como parámetro de búsqueda estas columnas. Y específicamente para esta iteración se usaron índices en la FECHAEJECUCION de la tabla ejecutaron.

Pero además hicimos 2 métodos en la clase DAO que nos permiten crear y eliminar índices cuando sea necesario.

Los índices creados de forma automática por Oracle son:



Los índices creados son las primary key de las tablas y ayudan al rendimiento en la medida de que una búsqueda sobre las tuplas de la tabla se hace buscando por las columnas que la diferencian, que justamente son las primary key.

2.2. Documentación del análisis

2.2.1. Documentación del escenario de pruebas

RFC8. CONSULTAR EJECUCIÓN DE ETAPAS DE PRODUCCIÓN 1

Consulta SQL

SELECT * FROM Ejecutaron e NATURAL INNER JOIN EstacionesProduccion NATURAL INNER JOIN EtapasProduccion WHERE e.FechaEjecucion BETWEEN TO_DATE(<<fecha #1>>,'YYYY-MM-DD') AND TO_DATE(<<fecha #2>>,'YYYY-MM-DD');

Distribución de los datos

Se han ingresado en la base de datos la información de 10000 pedidos. Estos pedidos aleatoriamente fueron entregados o no; aproximadamente el 50% de estos fueron entregados. Por cada pedido entregado se generaron 10 entradas en la tabla Ejecutaron, dado que cada producto en la base de datos requiere de 10 pasos aleatorios entre 10000 posibles etapas de producción para ser completado. Cada una de estas ejecuciones fue realizada en una estación de producción aleatoria entre 10000 posibles estaciones. Las ejecuciones fueron también realizadas por un empleado aleatorio entre 100, en un tiempo aleatorio de ejecución, en un rango de fechas de 6 meses (entre Junio y Diciembre del 2015).

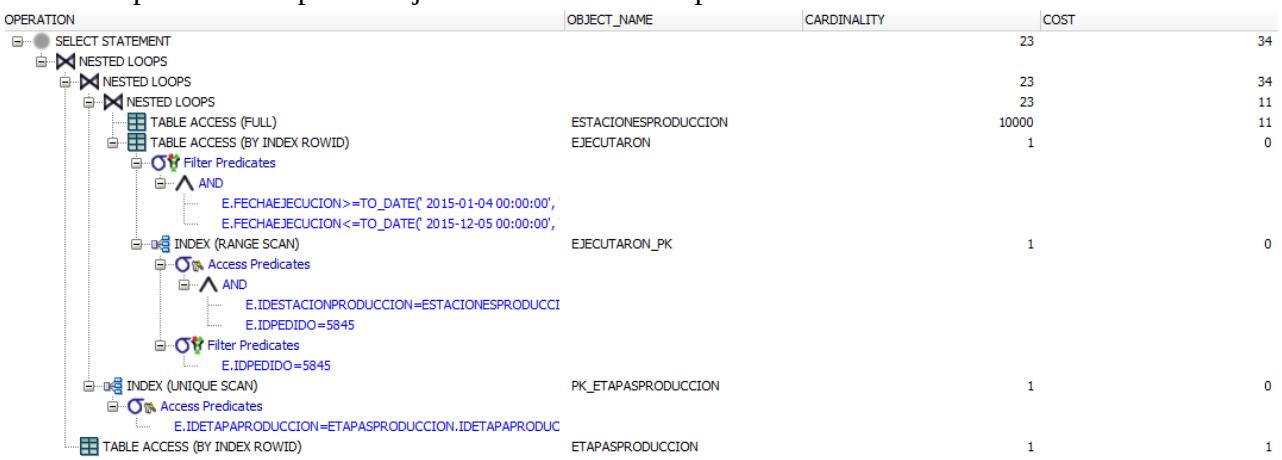
Se espera pues que aproximadamente, para un día dado dentro del rango, se hayan realizado 275 ejecuciones. Esto corresponde al 0.55% de ejecuciones

totales. Si se agregan parámetros adicionales a la búsqueda, el porcentaje de ejecuciones recuperadas cambian significativamente. Por ejemplo, si se restringen las ejecuciones a las correspondientes a un pedido específico se recuperaría el 0.02% de las ejecuciones.

Plan de ejecución y ejecución

En el plan de ejecución, se utiliza el índice de las llaves primarias de ambas tablas para agilizar el proceso de JOIN entre las tablas. Ante la consulta de las ejecuciones entre el 4 y 5 de julio de 2015 se obtuvieron 294 ejecuciones en 0.788 segundos. Al utilizar un índice en la fecha de ejecución se reduce el tiempo de ejecución a 0.386 segundos. Ante esta consulta no es necesario agregar más índices además del posible índice en fecha de ejecución.

Si se realiza la consulta filtrando por id de un pedido, se logra obtener las filas requeridas en 0.103 segundos para un pedido aleatorio elegido. A continuación se puede ver el plan de ejecución obtenido al aplicar el filtro.



Se aprecia que se hace uso del índice correspondiente a la llave primaria en la tabla correspondiente al id del pedido para agilizar la respuesta.

RFC9. CONSULTAR EJECUCIÓN DE ETAPAS DE PRODUCCIÓN 2

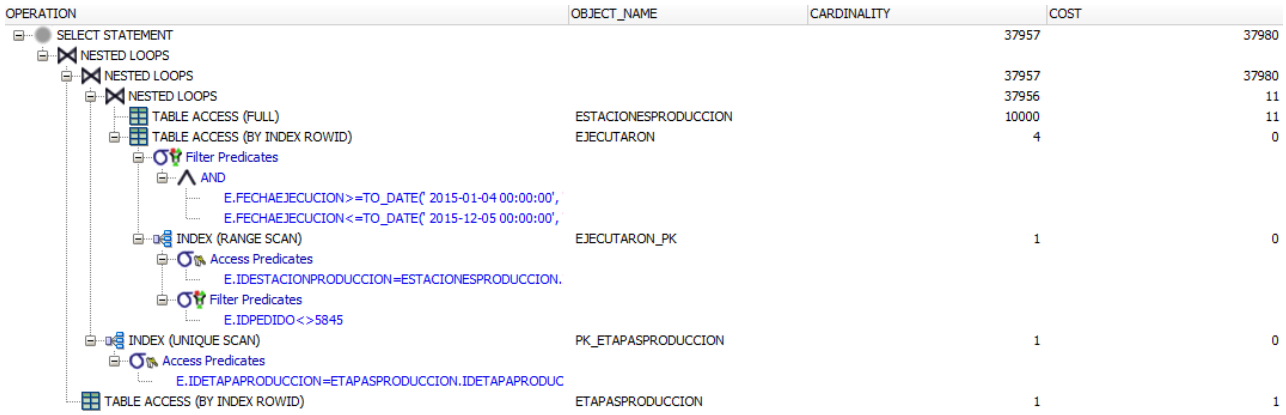
Consulta SQL

SELECT * FROM Ejecutaron e NATURAL INNER JOIN EstacionesProduccion NATURAL INNER JOIN EtapasProduccion WHERE e.FechaEjecucion BETWEEN TO_DATE(<<fecha #1>>,'YYYY-MM-DD') AND TO_DATE(<<fecha #2>>,'YYYY-MM-DD') AND e.idPedido != <<idPedido>>;

Distribución de los datos

La cantidad y distribución de datos es la misma que la asignada para la consulta anterior. Debe tenerse en cuenta que en este caso la selectividad puede ser bastante baja. Por ejemplo, si se niega la búsqueda por un id de pedido, se obtiene el 99.98% de las ejecuciones totales, lo cual constituye una baja selectividad.

Plan de ejecución y ejecución



En el presente plan de ejecución se aprecia que se hace uso de los índices aplicados por Oracle a las llaves primarias de las tablas. Se tiene además que el costo de ejecución viene principalmente del proceso de JOIN entre las tablas involucradas en la consulta. Esto se debe a la gran cantidad de datos que deben ser unidos en este caso.

En cuanto al tiempo de ejecución, se tiene que 1500 filas pueden ser recuperadas en 1.045 segundos. Dado que SQL Developer realiza paginación de sus resultados, se espera que, sabiendo que los resultados serían 49990 filas, para obtener todos los resultados se tendría que esperar 33.3 segundos. No debe olvidarse que una consulta de este tipo no tiene mucho sentido, y no arroja información relevante para la aplicación.

RFC10. CONSULTAR PEDIDOS 2

Consulta SQL

SELECT * FROM Pedidos NATURAL INNER JOIN Compran NATURAL INNER JOIN Productos NATURAL INNER JOIN (SELECT DISTINCT idProducto FROM EtapasProduccionProducto NATURAL INNER JOIN Requieren NATURAL INNER JOIN Recursos r WHERE r.costo><<costo>> AND r.tipoRecurso='<<tipo Recurso>>');

Distribución de los datos

Se han ingresado a la base de datos la información de 10000 recursos. A cada una de las 10000 etapas de producción que se encontraban en el programa se le agregaron además la necesidad de dos recursos aleatorios entre los presentes para su ejecución. El costo de los recursos agregados fue aleatorio, entre $10 y $500. Se debe resaltar que a cada recurso se le asignó un tipo aleatorio, por lo que aproximadamente 50% de estos son materias primas y el otro 50% de recursos corresponde componentes.

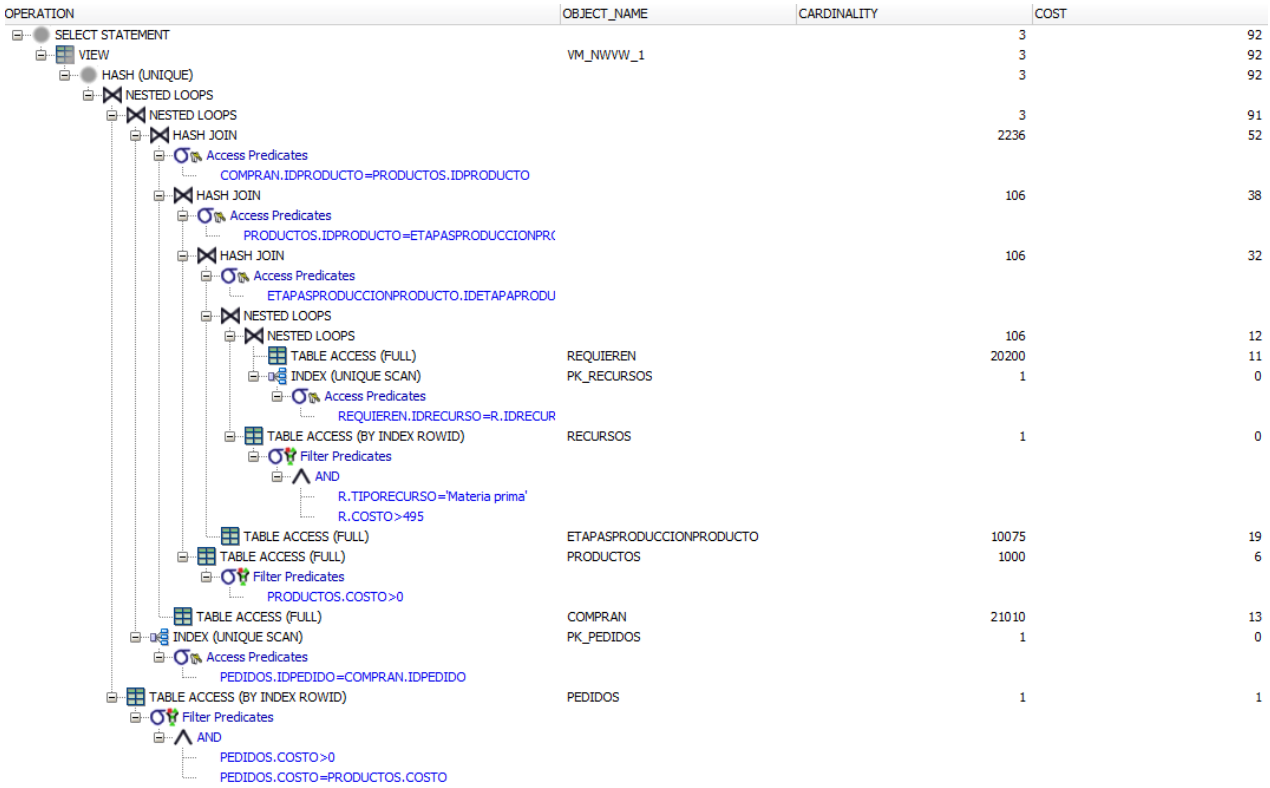
Ante la consulta planteada es un poco complicado calcular la selectividad. Si se toma un costo de 10 como parámetro la selectividad sería de 50%, si se

toma un costo de aproximadamente 260 la selectividad sería de 25%. Se tiene pues que la selectividad depende únicamente del costo seleccionado para la consulta.

Plan de ejecución y ejecución

El plan de ejecución aparece sumamente complicado, dado que consiste en gran parte de JOIN, puede apreciarse que buena cantidad de JOINS consisten en Hash Join. La ejecución de esta consulta, que para un costo con selectividad alta (495 de costo) arroja como resultado 146 tuplas, toma 1.05 segundos en ejecución.

Debe notarse que no hay muchas más posibilidades de variación a través del uso de índices, dado que la mayoría de JOINS ya se hacen sobre columnas que tienen índices sobre llaves primarias y, sobre el tipo de recurso no vale la pena tener un índice sobre el tipo de recurso dado que la selectividad es de 50% bajo este caso.



RFC11. CONSULTAR MATERIAL 2

Consulta SQL

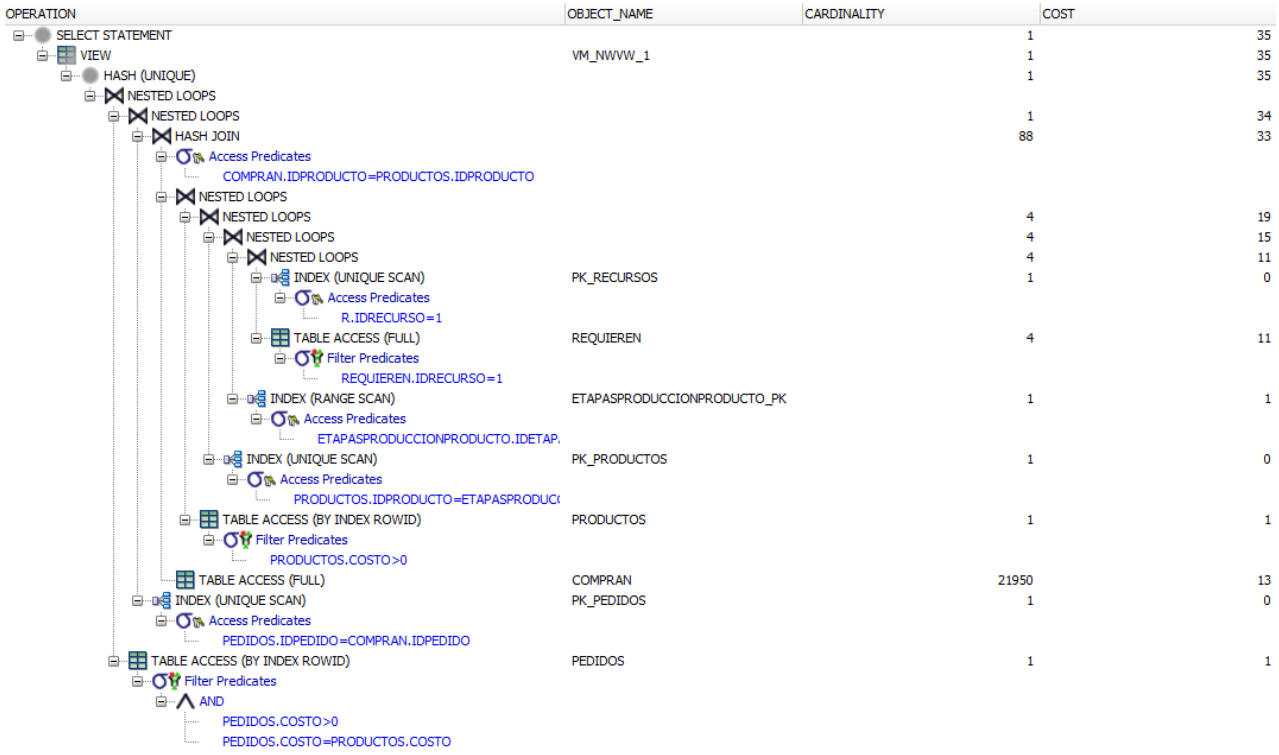
SELECT * FROM Pedidos NATURAL INNER JOIN Compran NATURAL INNER JOIN Productos NATURAL INNER JOIN (SELECT DISTINCT idProducto FROM EtapasProduccionProducto NATURAL INNER JOIN Requieren NATURAL INNER JOIN Recursos r WHERE r.idRecurso=<<idRecurso>>);

Distribución de los datos

La distribución de datos es la misma que la del requerimiento anterior. Se tiene que en este caso la selectividad es distinta. Si se trata de encontrar todos los pedidos en los que un recurso dado se encuentra se tiene que probablemente se seleccione 2 pedidos entre 10000, dada la forma en la que los datos fueron agregados en la base de datos.

Plan de ejecución y ejecución

El plan de ejecución es en buena parte parecido al del anterior requerimiento, dado que los JOIN realizados son básicamente los mismos. En esta consulta los índices de las llaves primarias son utilizados más dado que, además de los JOIN, resultan útiles en el filtro de la consulta. Debe destacarse también que en este caso no hay necesidad de agregar índices adicionales en alguna columna. La ejecución de esta consulta, para un id de recurso dado que resulta en 5 tuplas toma 1.957 segundos.



2.2.2. Análisis de eficiencia

- **RFC8. Consultar ejecución de etapas de producción 1 y 2**
  - **Escenarios con diferentes selectividades**

    La selectividad de este requerimiento se puede dividir en la selectividad de las fechas y la selectividad del criterio de búsqueda asociado. Por lo tanto los escenarios serían:

    Hay una dispersión muy baja de las fechas, todas son de un mismo periodo de tiempo, esto implicaría que si pido ese rango de fechas obtendré una gran cantidad de valores. Una selectividad muy baja.

    Hay una dispersión muy alta de las fechas, todas son de diferentes periodos de tiempo, esto implicaría que si pido ese rango de fechas obtendré una pequeña cantidad de valores. Una selectividad alta.

Los otros escenarios se referirían a la selectividad del criterio de búsqueda, en el caso de que el criterio se cumpla por todas las tuplas de la tabla la selectividad sería muy baja para la búsqueda con correspondencia y muy alta para la búsqueda sin correspondencia. Mientras que si el criterio solo lo cumplen algunas tuplas de la tabla la selectividad sería muy alta para la búsqueda con correspondencia y muy baja para la búsqueda sin correspondencia.

- **Diseño de un plan de ejecución**

SELECT * FROM Ejecutaron e NATURAL INNER JOIN EstacionesProduccion NATURAL INNER JOIN EtapasProduccion WHERE e.FechaEjecucion BETWEEN TO_DATE(<<fecha #1>>,'YYYY-MM-DD') AND TO_DATE(<<fecha #2>>,'YYYY-MM-DD');

Lo que esperaríamos es que la base de datos hiciera primero la selección solo de aquellas tuplas que cumplen con las condiciones de fecha y otros parámetros indicados y luego hiciera el JOIN por la columna que tienen en común las tablas.

- **Comparar con el de Oracle**

Lo primero que hace Oracle es seleccionar las tuplas de la primera tabla que tienen una correspondencia con la segunda tabla. Como pensábamos lo segundo que hace es seleccionar las tuplas que cumplen con la restricción buscada, no espera hasta tener todas para hacer la selección de las que quiere. Esto seguramente es más eficiente que esperar a tener todas para seleccionar, porque implica menos operaciones de conexión de tuplas.

- **RFC10. Consultar pedidos 2**
  - **Escenarios con diferentes selectividades**

La selectividad de la consulta depende de la selectividad de tipo de material y del costo.

Para el caso de esta iteración nosotros planteamos una selectividad de 50%, lo que es una selectividad baja. Pero se puede pensar en otros casos donde hayan más tipos de materiales y estén distribuidos equitativamente, lo que implica una selectividad alta.

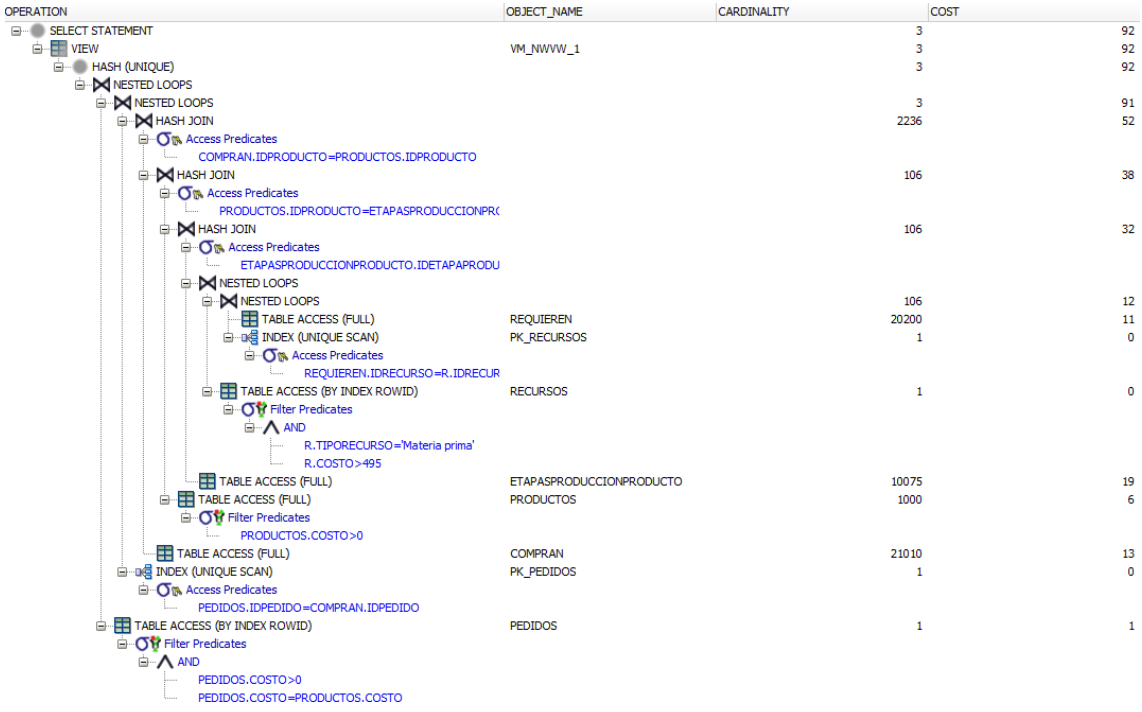
Respecto al costo los posibles escenarios serían donde hay una distribución uniforme que implica que si buscamos costos mayores a un costo pequeño encontrare muchos resultados, baja selectividad, en cambio sí buscamos costos mayores a un costo muy alto encontraremos pocos resultados, alta selectividad.

- **Diseño de un plan de ejecución**

SELECT * FROM Pedidos NATURAL INNER JOIN Compran NATURAL INNER JOIN Productos NATURAL INNER JOIN (SELECT DISTINCT idProducto FROM EtapasProduccionProducto NATURAL INNER JOIN Requieren NATURAL INNER JOIN Recursos r WHERE r.costo><<costo>> AND r.tipoRecurso='<<tipo Recurso>>');

Lo que esperaríamos que la base de datos hiciera es que primero en el select interno seleccionara las tuplas que cumplen con los criterios y luego el join de las dostablas, luego se esperaría que se hiciera el join con las demás tablas.

- **Comparar con el de Oracle**



Justamente como en la consulta anterior lo que hace Oracle es hacer el filtro de las tuplas que cumplen con las condiciones en la primera tabla para luego hacer joins con las otras tablas.

- **RFC11. Consultar material 2**
  - **Escenarios con diferentes selectividades**

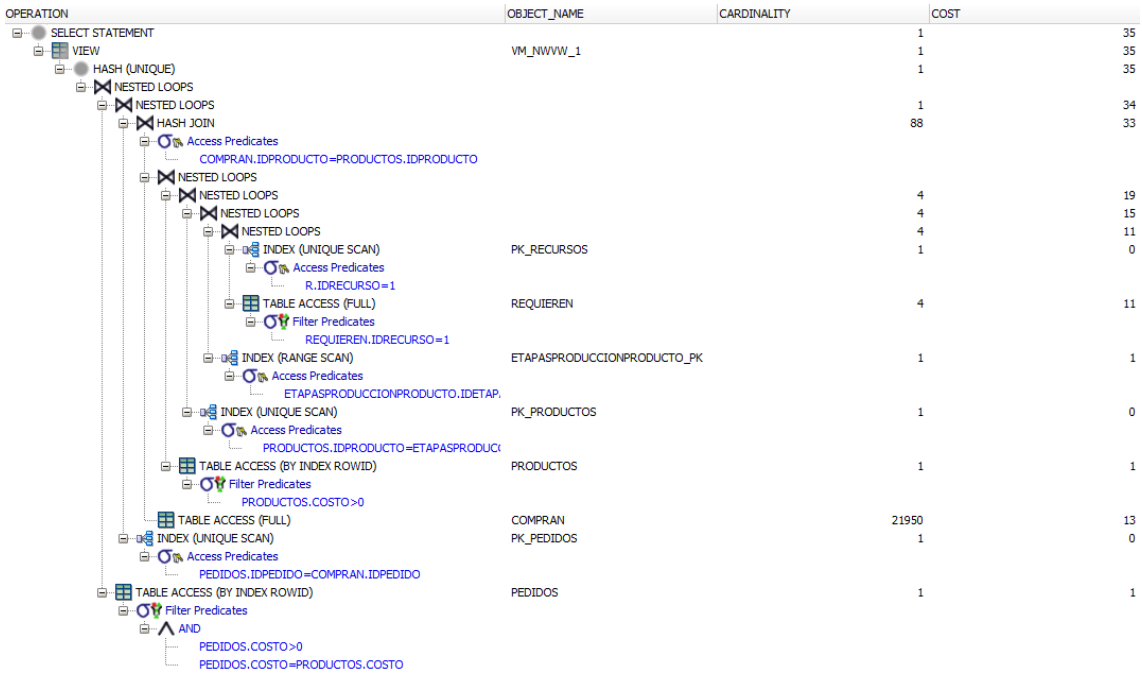
La selectividad en este caso depende únicamente del porcentaje de pedidos de la empresa que representan los pedidos en los que haya estado involucrado el material solicitado. Si el material ha estado en la mayoría de pedidos que le han realizado a la empresa la selectividad va a ser muy baja, mientras que si es un material con pocos pedidos con respecto a los otros materiales la selectividad va a ser muy alta.

- **Diseño de un plan de ejecución**

SELECT * FROM Pedidos NATURAL INNER JOIN Compran NATURAL INNER JOIN Productos NATURAL INNER JOIN (SELECT DISTINCT idProducto FROM EtapasProduccionProducto NATURAL INNER JOIN Requieren NATURAL INNER JOIN Recursos r WHERE r.idRecurso=<<idRecurso>>);

Lo que esperaríamos es que Oracle seleccionara de la tabla recursos la tupla con el id deseado y le hiciera Join con todas las tuplas de la tabla REQUIEREN, únicamente para encontrar los productos en los que participa el recurso. Luego se harían los Joins con las otras tablas para encontrar la información de los pedidos en los que están estos productos.

- **Comparar con el de Oracle**



Justo como se pensaba lo primero que hace es encontrar el recurso específico que busca para luego hacer los Joins para encontrar la información que quiere. El plan de ejecución es en buena parte parecido al del anterior requerimiento, dado que los JOIN realizados son básicamente los mismos. En esta consulta los índices de las llaves primarias son utilizados más dado que, además de los JOIN, resultan útiles en el filtro de la consulta.

3. Construcción de la aplicación y análisis de resultados

3.1. Documentación del proceso de la carga de datos

El proceso de carga de datos fue realizado en a través de una conexión generada desde eclipse. En java se escribieron métodos diseñados para generar querys de inserción con información aleatoria que fueron ejecutados en un ciclo. Cada cierta cantidad de inserciones un commit era realizado. El proceso de inserción se realizó de tal forma que en todo momento la base de datos mantuviera coherencia. Si en algún momento una inserción fallaba, se ejecutaba un rollback con el fin de que la información entre tablas no llegara a ser inconsistente. Aproximadamente 8 métodos distintos fueron utilizados en el proceso de generación de datos. Todos los métodos fueron ejecutados sin el despliegue de la aplicación en JBoss.

3.2. Análisis del proceso de optimización y el modelo de ejecución de consultas

La diferencia principal es que el manejador de bases de datos tiene desarrollada una serie de algoritmos que le permiten encontrar de forma muy eficiente los datos, así como una forma de organizar físicamente la información para encontrarla de forma más efectiva, reduciendo los tiempos de la consulta a tiempos muy cortos. Habitualmente tienen los datos organizados en arboles b+ o tablas hashing expandibles, lo que les permite hacer operaciones de selección de una manera muy rápida. Además normalmente cuando uno hace la consulta el manejador no carga todos los datos inmediatamente sino que a medida que uno va pidiendo más datos él los va buscando y trayendo a la memoria principal.

Mientras que si uno quisiera hacer esto en la aplicación le tomaría mucho tiempo, puesto que si quisiera primero organizar los datos para hacer búsquedas sobre ellos tendría que sumarle al tiempo de búsqueda el tiempo de organizar los datos y para cada búsqueda con datos distintos tendría que volver a hacer el mismo proceso. O si intentara hacer la búsqueda sin ordenar el tiempo sería que para cada operación tendría que recorrer todos los datos que tiene.