**Iteración 4**

Julián Andrés Bermúdez Valderrama, Diego Francisco Sanabria Dimate

Universidad de los Andes, Bogotá, Colombia

{ja.bermudez10, df.sanabria761}@uniandes.edu.co

Fecha de presentación: mayo 07 de 2017

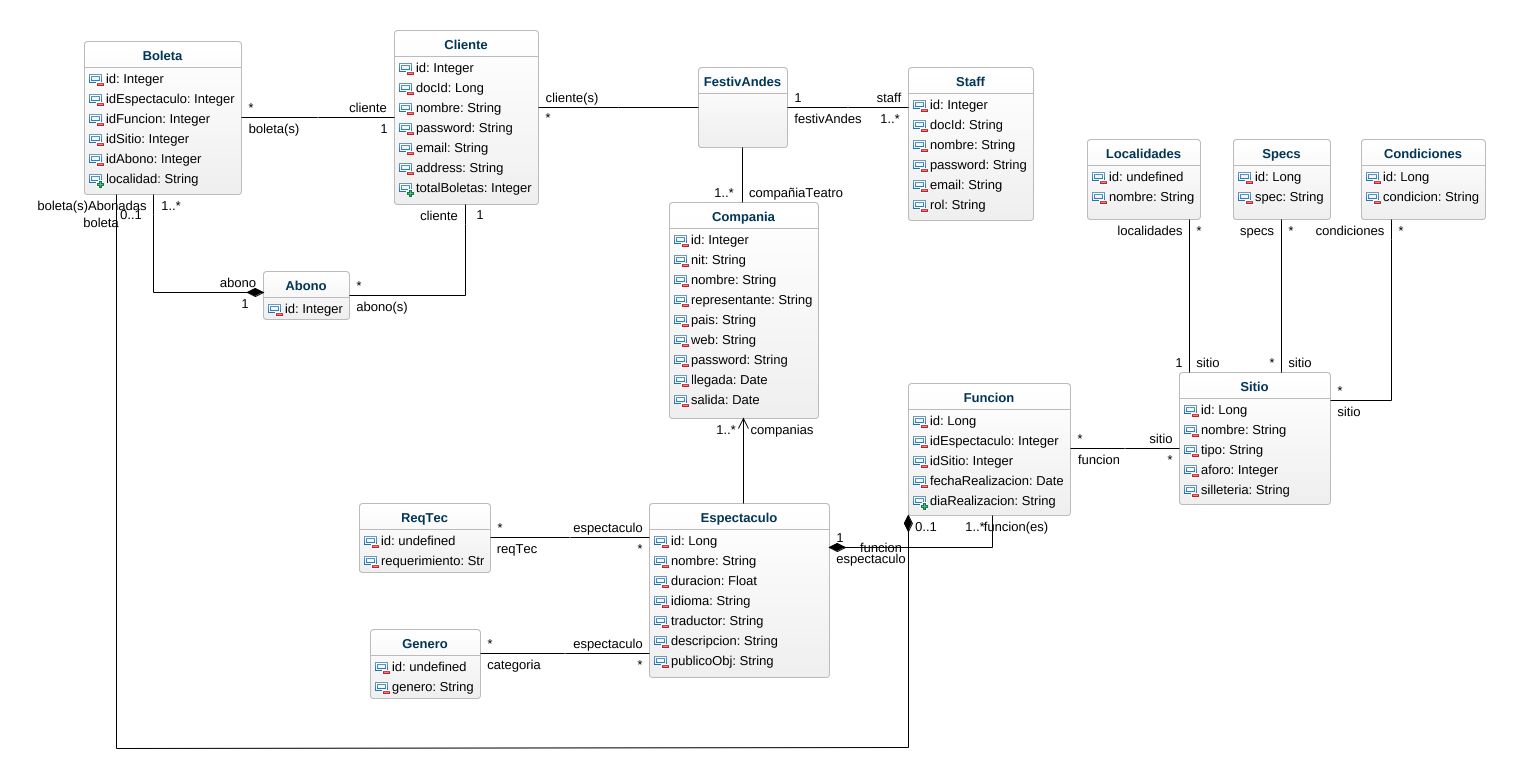
1. Ajustes al modelo UML

Imagen 1

En la imagen 1 se aprecia el nuevo modelo UML, modificado de acuerdo a los nuevos requerimientos funcionales para FestivAndes. Para esto se implementó una conexión entre boleta y función, como fue sugerida por el monitor, de manera tal que exista una tabla vinculante sobre la cual realizar las consultas. Se agregó además un nuevo atributo para los clientes, el cual modela el total de boletas adquiridas por el mismo, esto con el fin de solventar el requerimiento funcional de consulta 11.

Fue necesario también adicionar un atributo que modele el día de realización de una función de manera tal que se facilite el indexa-miento y posterior filtro de consulta sobre esto, por este mismo motivo se agregó una localidad directamente en la boleta, así la cantidad de tablas necesarias para resolver el requerimiento funcional de consulta 12 sea la mínima posible.

1. **Ajustes al modelo relacional**
   1. **Selección de índices**

Para el requerimiento funcional de consulta 11 se optó por utilizar tres Índices secundarios de tipo BITMAP, día de la semana (Sobre la tabla *Función*), elementos del escenario (Sobre la tabla de especificaciones por sitio) y por tipo de localidad (Sobre la tabla *Boletas*), este último con el fin de satisfacer además el requerimiento de tiempo de consulta para el requerimiento funcional 12. Se seleccionaron así dado que con este tipo de índice las consultas por atributos específicos y poco polivalentes minimizan su tiempo de ejecución, ya que no se esperan diversos elementos sobre localidad, y mucho menos sobre día de la semana, éste es el índice idóneo para esta consulta.

Además, se optó por implementar un índice secundario de tipo Árbol B+ sobre la fecha de realización, esto puesto que el requerimiento exige consultas sobre rango de fechas y esta estructura de índice es la única que solventa de manera efectiva los problemas de rangos.

Por otro lado, para el requerimiento funcional de consulta 12 se aprovechará el índice secundario sobre el atributo localidad en la clase *Boletas* y además se indexará mediante un árbol B+ secundario a los clientes por total de boletas compradas.

**Documentación del escenario de pruebas:**

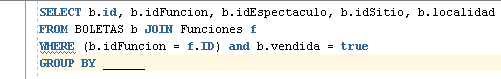
Para la realización de las pruebas respectivas se usó *Mockaroo*, página con la cual creamos 100 clientes, 10 compañías, 200 espectáculos, 2000 boletas y 10 funciones por espectáculo. Esto con el fin de tener una base de datos densamente poblada para probar, con datos aleatorios, el funcionamiento de nuestros índices aplicados.

**-Sentencias SQL:**

**RFC 9:**

**RFC 10­**

**RFC 11:**

****

**RFC 12:**

**Distribución de datos:**

Los datos utilizados para las pruebas fueron de carácter aleatorio, así pues, al usar las sentencias por criterio la distribución fue uniformemente dispersa, dado que *Mockaroo* no repitió discriminadamente datos. De igual manera hubo cierta cantidad de funciones con muy pocos clientes, mientras habían funciones completamente opuestas.

Se usaron como parámetros para los primeros requerimientos los valores de: Nombre y Total de boletas, con el primero la consulta ejecuta una lectura total de la tabla, siendo este más demorado demostrando como la falta de orden afecta la consulta, por el tiempo que demora en encontrar los datos en disco.

Al usar el segundo parámetro, aunque no fuera clave para la consulta realizada, las búsquedas demoraron mucho menos, ya que SQLDeveloper hacía uso del índice para agilizar sus búsquedas.

1. **Análisis de resultados**

Al optimizar las consultas, generar índices diversos y delegarle la ejecución de las consultas a Oracle la velocidad de procesamiento es mucho más alta. No siendo esto suficiente al no usar instrucciones de control en memoria principal la cantidad de accesos al disco disminuye y aumenta el volumen de datos que faciliten las consultas. Todo lo anterior implica un desempeño radicalmente mejor en cada aspecto.

De ser usada una ejecución en memoria principal, sin los niveles de refinamiento del software usado en el curso, los tiempos crecerían drásticamente, además de lastimar la infraestructura del equipo dado que exigiría el máximo posible de los recursos, durante mucho tiempo. Esto es arriesgado y poco rentable, pues con tiempos tan elevados la calidad de la aplicación realizada sería muy baja y no otorgaría beneficios visibles.

La calidad del optimizador también permite que los datos lleguen limpios, intactos y en la menor cantidad de movimientos a disco posibles. Todo esto permite visualizar resultados idóneos a las consultas realizadas.