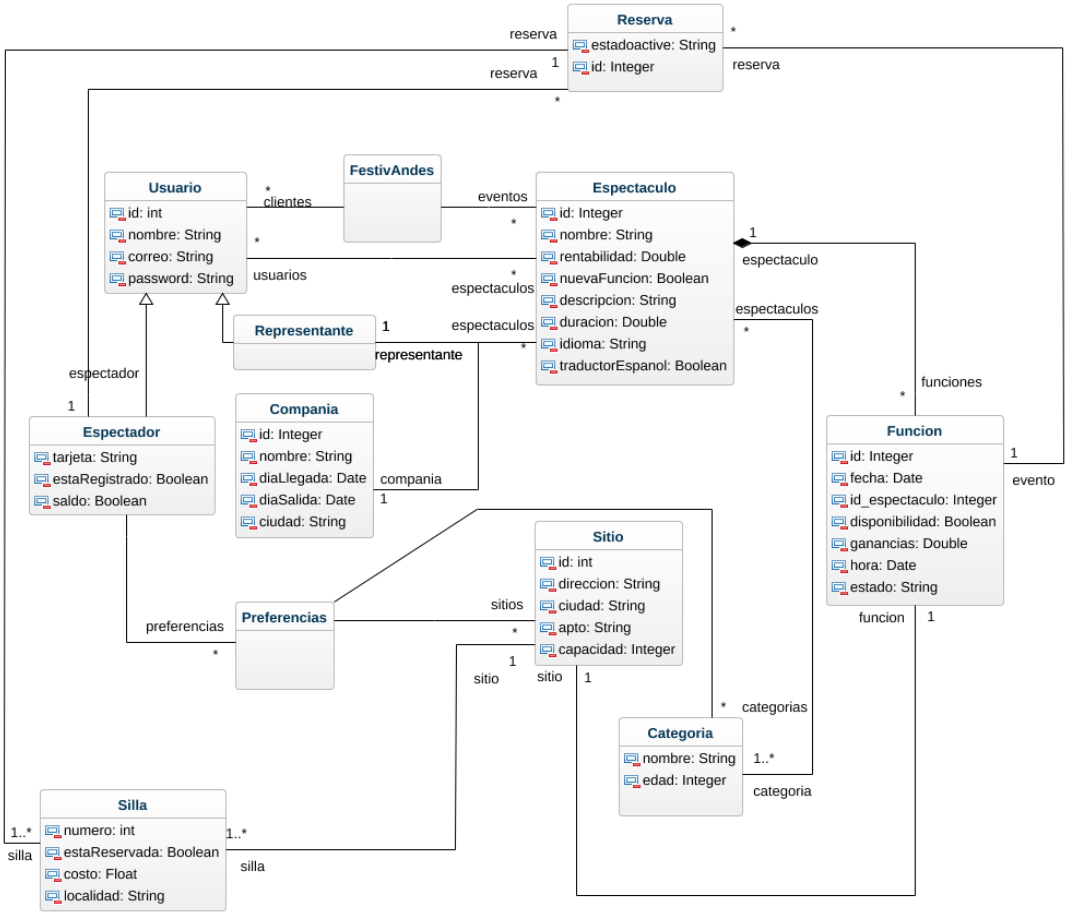
Iteración 3 - FestivAndes
Cambios del modelaje de la Iteración 2 y Análisis de Transaccionalidad

Paula J. Alvarado Zabala, Ana C. Fandiño de la Hoz
201313033, 201326407
Sistemas Transaccionales
Universidad de los Andes, Bogotá, Colombia
{pj.alvarado10, ac.fandino10}@uniandes.edu.co

Fecha de presentación: Abril 12 de 2017

Contenido

1. Análisis (UML)



En el Modelo Lógico de la iteración 2 a la iteración 3 se cambió muy poco. Lo más notorio es la adición del atributo estadoactive a la clase Reserva. Además, a la clase Reserva se le puso un id y se conectó con un usuario Espectador y una función. Y a la silla se le conectó con una reserva. Adicional a esto, por la retroalimentación de la sustentación pasada, se conectó la clase Representante directamente con el Espectáculo de su compañía y se considera la clase Representante como un usuario administrador. El Espectador ahora tiene un atributo de saldo y se eliminan las clases innecesarias como Banco. Finalmente se agrega la clase Preferencias para facilitar la realización de un requerimiento, la cual es dependiente de un espectador y posee sitios y categorías. La clase Evento cambia de nombre por Función, que es más entendible.

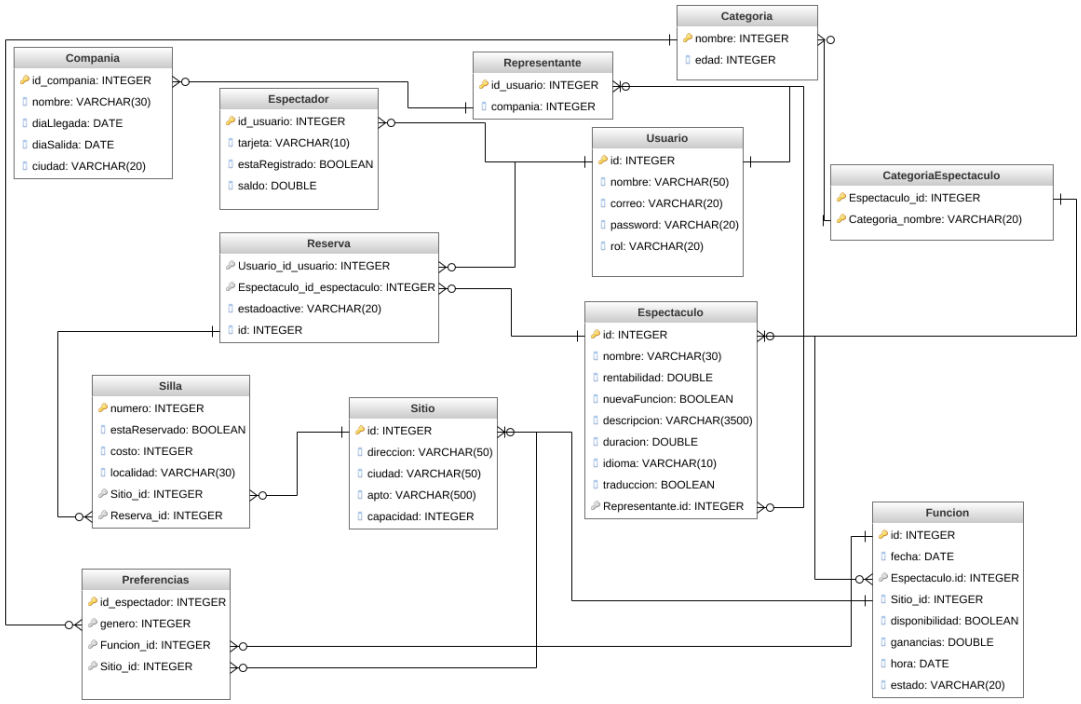
2. Diseño de la Aplicación

a. Impacto de los nuevos Requerimientos, Tablas y Modelo Relacional.

El impacto de los nuevos requerimientos en el modelo relacional ayuda a que el programa sea más eficiente pues la conexión directa de Representante con Espectáculo reduce el

tiempo que toma tener que pasar por un usuario. Todas las clases, al contener datos, son persistentes y les corresponde una tabla de la base de datos. Las tablas y el modelo relacional modificado se presentan a continuación:

a.1. Modelo Relacional



En este nuevo modelo relacional se ve que la tabla Reserva tiene dos nuevos atributos, uno es id que es llave primaria, y el otro es estadoactive, que es el atributo que va a contar si la reserva está activa o fue cancelada. También hay una nueva relación entre Representante y Espectáculo, en donde la tabla Espectáculo cuenta con un atributo que es llave foránea con el id del Representante. Además se agregó una nueva tabla llamada Preferencias.

a.2. Modelo de Tablas Relacional

Usuario				
id	nombre	Correo	password	rol
PK	NN	NN, ND	NN	NN

Espectador			
id_usuario	tarjeta	estaRegistrado	saldo
PK, FK(Usuario.id_usuario)	NN	NN	NN

Representante	
id_usuario	id_compania
PK, FK(Usuario.id_usuario)	NN, FK(Compania.id_compania)

Compania				
id	nombre	diaLlegada	diaSalida	ciudad
PK	NN	NN, CK(>hoy)	NN, CK(>diaLlegada)	NN

Reserva			
id_espectador	id_funcion	estadoactive	id
PK1, FK(Espectador.id_usuario)	PK2, FK(Funcion.id)	NN	PK

Espectaculo								
id	nombre	Rentabilidad	nuevaFuncion	descripcion	duracion	idioma	traduccion	id_representante
PK	NN	NN	CK	NN	NN	NN	NN	NN,FK(Representante.id_usuario)

Funcion								
id	fecha	id_espectaculo	id_sitio	disponibilidad	ganancias	duracion	hora	estado
PK	NN, CK(>Hoy)	NN, FK(Espectaculo.id)	NN, FK(Sitio.id)	NN	NN	NN	NN	NN

Sitio				
id	direccion	ciudad	apto	capacidad
PK	NN,ND	NN	NN	NN

Silla					
numero	estaReservada	costo	localidad	id_sitio	id_reserva
PK	NN	CK(⩾0)	NN	FK(Sitio.id), NN	FK(Reserva.id)

Categoria	
nombre	edad
PK	NN

CategoriaEspectaculo	
id_Evento	nombre_Categoria
PK1, FK(Evento.id_evento)	PK2, FK(Categoria.nombre)

Preferencias			
id_espectador	genero	id_funcion	id_sitio
PK, FK(Espectador.id)	FK(Categoria.id)	FK(Funcion.id)	FK(Sitio.id)

a.3. Toma de decisiones e impacto transaccionalidad

En cuanto al requerimiento no funcional de transaccionalidad, se decidió hacer la solución de aislamiento por serialización con un control pesimista, es decir, con candados exclusivos de escritura para que cuando un usuario acceda a los datos para escribir, ningún otro usuario tenga acceso a los datos (ni lectura ni escritura) hasta que se finalice la escritura. Mientras tanto, si no hay usuarios escribiendo, se tiene permitido que varios usuarios accedan a los datos para lectura. Esto evita problemas de concurrencia (lecturas sucias o inconsistentes, fantasmas, abrazos mortales y hambre) y maneja la transaccionalidad de una manera limpia y sencilla.

a.4. Tablas Consulta SQL

A su vez, se muestra el listado de tablas ahora obtenido mediante una búsqueda SQL:

Consulta SQL:

```
Select table_name, column_name, constraint_name from
user_cons_columns where owner='ISIS2304B071710' order by
table_name, column_name;
```

Tabla de Resultado:

TABLE_NAME	COLUMN_NAME	CONSTRAINT_NAME
CATEGORIAESPECTACULO	ID_ESPECTACULO	SYS_C00100548
CATEGORIAESPECTACULO	ID_ESPECTACULO	CATEGORIAESPECTACULO_PK
CATEGORIAESPECTACULO	NOMBRECATEGORIA	SYS_C00100547
CATEGORIAESPECTACULO	NOMBRECATEGORIA	CATEGORIAESPECTACULO_PK
CATEGORIAS	NOMBRE	SYS_C00100651
CATEGORIAS	NOMBRE	CATEGORIAS_PK
COMPANIAS	ID	SYS_C0097281
COMPANIAS	ID	COMPANIA_PK
ESPECTACULOS	ID	ESPECTACULO_PK
ESPECTACULOS	ID	SYS_C00100670
ESPECTADORES	ID	SYS_C00101168
ESPECTADORES	ID	ESPECTADORES_PK
FUNCIONES	ID	EVENTOS_PK
FUNCIONES	ID	SYS_C00101486
PREFERENCIAS	ID_ESPECTADOR	SYS_C00101795
PREFERENCIAS	ID_ESPECTADOR	PREFERENCIAS_PK
REPRESENTANTES	ID	ID
REPRESENTANTES	ID	SYS_C00108902
REPRESENTANTES	ID	USUARIO_ID
RESERVAS	ID_ESPECTADOR	RESERVA_PK
RESERVAS	ID_ESPECTADOR	SYS_C00102120
RESERVAS	ID_FUNCION	SYS_C00102121
RESERVAS	ID_FUNCION	RESERVA_PK
SILLAS	NUMERO	SILLAS_PK
SILLAS	NUMERO	SYS_C00102164
SITIOS	ID	SYS_C00102166
SITIOS	ID	SITIOS_PK
USUARIOS	CORREO	SYS_C0097255
USUARIOS	ID	SYS_C0097254
USUARIOS	ID	USUARIO_PK

b. BCNF

Se valida que el modelo se encuentra en la forma normal de Boyce-Codd pues no hay dependencias ni trivialidades entre atributos primos. Además, se encuentra en 3FN, es decir que tampoco tiene dependencias transitivas entre atributos no primos de la relación, y tampoco hay dependencias parciales (un no primo depende de un primo que es subconjunto de una llave). Se explicará con más detalle, tabla por tabla, a continuación:

Tabla Usuario	
Llave	Id_usuario
Atributos primos	Id_usuario
Atributos no primos:	Nombre, correo, password,rol
Dependencia parcial	No hay pues de por sí, la llave es monoatributo.
Dependencia transitiva	No hay pues nombre no determina correo ni viceversa, ni nombre determina password ni viceversa, y el correo no determina password ni viceversa.
Trivialidades	No hay pues la llave es monoatributo

Tabla Espectador	
Llave	Id_usuario
Atributos primos	Id_usuario
Atributos no primos:	Tarjeta, estaRegistrado, saldo
Dependencia parcial	No hay parciales pues de por sí, la llave es monoatributo.

Dependencia transitiva	No hay pues tarjeta no determina si estaRegistrado ni viceversa
Triviliadedes	No hay pues la llave es monoatributo

Tabla Representante	
Llave	Id_usuario
Atributos primos	Id_usuario
Atributos no primos:	Id_compania
Dependencia parcial	No hay parciales pues de por sí, la llave es monoatributo.
Dependencia transitiva	No hay pues solo hay un atributo no primo que depende de uno primo que es llave
Triviliadedes	No hay pues la llave es monoatributo

Tabla Compania	
Llave	Id_compania
Atributos primos	Id_compania
Atributos no primos:	nombre, diaLlegada, diaSalida, ciudad
Dependencia parcial	No hay parciales pues de por sí, la llave es monoatributo.
Dependencia transitiva	No hay pues nombre no determina el dia de llegada, ni el dia de salida, ni la ciudad y viceversa. Y dia llegada no determina el dia de salida ni la ciudad ni viceversa. Ni la ciudad determina el dia de salida ni viceversa.
Triviliadedes	No hay pues la llave es monoatributo

Tabla Reserva	
Llave	{id}
Atributos primos	id
Atributos no primos:	estadoactive, Id_usuario, id_funcion,
Dependencia parcial	No hay parciales pues la llave es monoatributo
Dependencia transitiva	No hay pues ningun no primo depende de otro.
Triviliadedes	No hay pues la llave es monoatributo

Tabla Espectáculo	
Llave	Id_espectaculo
Atributos primos	Id_espectaculo
Atributos no primos:	Nombre, rentabilidad, nuevaFuncion, descripción, duración, idioma, traduccionEspanol, id_representante
Dependencia parcial	No hay parciales pues de por sí, la llave es monoatributo.

Dependencia transitiva	No hay pues ningún atributo no primo está determinado por otro no primo.
Triviliadedes	No hay pues la llave es monoatributo

Tabla Funcion	
Llave	Id_funcion
Atributos primos	Id_funcion
Atributos no primos:	Fecha, id_espectaculo, disponibilidad, id_sitio, ganancias, hora, estado
Dependencia parcial	No hay parciales pues de por sí, la llave es monoatributo.
Dependencia transitiva	No hay pues ningún atributo no primo está determinado por otro no primo.
Triviliadedes	No hay pues la llave es monoatributo

Tabla Sitio	
Llave	Id_sitio
Atributos primos	Id_sitio
Atributos no primos:	Dirección, ciudad, apto, capacidad
Dependencia parcial	No hay parciales pues de por sí, la llave es monoatributo.
Dependencia transitiva	No hay pues ningún atributo no primo está determinado por otro no primo.
Triviliadedes	No hay pues la llave es monoatributo

Tabla Silla	
Llave	Numero
Atributos primos	Numero
Atributos no primos:	estaReservada, costo, localidad, eventoReserva, id_sitio, id_reserva
Dependencia parcial	No hay parciales pues de por sí, la llave es monoatributo.
Dependencia transitiva	No hay pues ningún atributo no primo está determinado por otro no primo.
Triviliadedes	No hay pues la llave es monoatributo.

Tabla Categoria	
Llave	Nombre
Atributos primos	Nombre
Atributos no primos:	Edad
Dependencia parcial	No hay parciales pues de por sí, la llave es monoatributo.
Dependencia transitiva	No hay pues solo hay un atributo no primo.
Triviliadedes	No hay pues la llave es monoatributo.

Tabla CategoriaEspectáculo	
Llave	{Id_espectaculo, nombre_categoria}
Atributos primos	Id_espectaculo, nombre_categoria
Atributos no primos:	-----
Dependencia parcial	No hay parciales pues no hay atributos no primos.
Dependencia transitiva	No hay pues no hay atributos no primos.
Triviliadedes	No hay pues el espectáculo no determina la categoría ni viceversa.

Se valida que el modelo no presenta anomalías de inserción, borrado o actualización con respecto a las reglas de negocio pues los requerimientos tienen un nivel de aislamiento transaccional (serializable), que garantiza el cumplimiento de las propiedades ACID para cada transacción realizada por estos requerimientos y no permite que la transacción tenga actualizaciones intermedias ya que reserva la escritura y lectura para el primer usuario que acceda a la información, solucionando problemas de los abrazos mortales y hambre.

c. Lógica de los nuevos requerimientos

- RF10: Es igual al RF8, la única diferencia es que ahora se puede reservar más de una silla al tiempo. Para esto se utilizó el método de RF8 varias veces dentro del método RF10 para reservar varias sillas al tiempo, dichas sillas una al lado de la otra. Como mecanismo para garantizar las propiedades ACID, se utilizó una sentencia que cambie el estado de la silla a reservada y un candado que permita que cuando una persona reserva en una función, nadie más puede reservar en esa función, dicha sentencia se encuentra en el RF8 pues RF10 llama a este para funcionar.

- RF11: Al igual que RF8, pero en vez de pagar la boleta, solo la reserva con un abono del 20% del valor de la boleta (si quedan 3 semanas para la función y hay disponibilidad), y cuando se va a pagar, se le descuentan 20% del valor de la boleta (es decir, se suma a las ganancias el valor de la boleta menos el 40%, 20% que ya pagó, y 20% de descuento). Para implementar la lógica se usó una condición (if) que comprobara que había disponibilidad en el evento, la silla estuviera libre y que la fecha actual fuera 3 semanas antes de la función, y para cuando fuera a pagar se hizo una condición que mirara si el usuario tenía el atributo abono mayor a 0. Como se utiliza RF8, se debe restar el valor de la boleta en las ganancias y sumarle solo el 20% del abono. Para garantizar las propiedades ACID, se utilizaron los mismos mecanismos que RF10 usados en RF8.

- RF12: Este requerimiento es la implementación inversa de RF8, por ende, se utilizó el método deleteReserva para eliminar todas las reservas y liberar las sillas, pero además de eso se descontó de las ganancias el costo de esa reserva. . Para garantizar las propiedades ACID, se utilizaron los mismos mecanismos usados en RF8 (candados

exclusivos) pero esta vez directamente en el método de deleteReserva, en este caso la reserva continúa en la tabla pero su estado pasa de activo a cancelado.

- RF13: En este requerimiento es igual que el RF12 pero en vez de una compra, de un abono. Se realiza el mismo procedimiento con el método deleteAbono, solo que esté en vez de eliminar una tupla Reserva, lo que hace es un update en el atributo abono de una tupla de Reserva, cambiando su valor a 0. Para garantizar las propiedades ACID, se utilizaron los mismos mecanismos usados en RF8 (candados exclusivos) pero esta vez directamente en el método de deleteAbono.

- RF14: Este requerimiento cancela una función, es decir, utiliza el método DAO de eliminarEvento y cuando lo realiza, también hace un llamado al requerimiento RF12 para cancelar todas las reservas de los asistentes, para ello, hace una iteración que va cancelando todas las reservas asociadas a ese evento. Para garantizar las propiedades ACID, se utilizaron los mismos mecanismos de candado exclusivo en eliminarEvento, y además en RF12 ya está implementado su propio mecanismo de concurrencia.

- RFC7: Este requerimiento consulta las funciones a las que asiste un cliente, buscando en la tabla de Reservas, las reservas con el id del cliente consultado, y recorre el id del evento para saber el nombre del espectáculo, mira el estado de la reserva y cuenta la cantidad de reservas canceladas de ese cliente. Como es un requerimiento de lectura, no tiene ningún tipo de candados, pero no devuelve ningún dato si se está modificando algo en la tabla de Reservas.

- RFC8: Este requerimiento consulta todos los datos de una compañía. Con el id de la compañía hace una consulta por todos los espectáculos y sus respectivas funciones (eventos) y calcula con las ganancias y la rentabilidad los datos solicitados. Se ordena según el espectáculo. Como es un requerimiento de lectura, se utilizan igual mecanismos que RFC7 para garantizar las propiedades ACID.

- RNF5: Se maneja la transaccionalidad de cada una de los requerimientos anteriores simplemente definiendo como mecanismo para garantizar las propiedades ACID su nivel de aislamiento como serializable pesimista.

3. Construcción de la Aplicación

a. Se ajustaron las tablas (véase en SQL Developer, usuario ISIS2304B071710).
b. Se poblaron las tablas (véase en SQL Developer).
c. El proyecto en eclipse con las nuevas modificaciones es este, este documento se encuentra en la carpeta Docs de dicho proyecto. Véase con mayor detenimiento con los pasos a seguir en el README.txt.
d. El archivo Test de Excel se encuentra adjunto en esta misma carpeta.

4. Bono

Diferencias del manejo transaccional entre un contenedor y un programador

La principal diferencia entre operar el manejo transaccional con el servidor de aplicaciones o que un programador lo haga manual, reside en que el contenedor de aplicaciones utiliza las configuraciones designadas por defecto para realizar cada una de las transacciones (autocommit, ningún savePoints, rollback hasta el inicio de la transacción sin guardar avances en la transacción y nivel de aislamiento en lectura confirmada). Mientras que, cuando es el programador quien debe hacerlo debe, debe determinar cada uno de las configuraciones para realizar la transacción.

LO BUENO

Contenedor de aplicaciones:

- Brinda un nivel de abstracción atractivo para el programador, quitándoles las tareas del manejo transaccional y permitiéndole concentrarse en las tareas de desarrollo a la vez que el servidor de aplicaciones le brinda el manejo transaccional que permitirá realizar la gran mayoría de transacciones.
- Las interfaces ofrecidas por el contenedor hacen que las transacciones sean fáciles de implementar y de utilizar. La escalabilidad, además, se hace más fácil.
- Como el contenedor es desarrollado alrededor de estándares pre-establecidos, sus módulos se acoplaran más fácilmente a ambientes extraños de despliegue.
- El manejo de la información es limpio y provee soporte de seguridad.

Programador:

- Permite que el programador especifique las características de la transacción, realizando las transacciones más precisas y como él prefiera, ajustándose a las reglas de negocio y a las necesidades particulares de un problema y del programador.
- Las transacciones pueden adaptarse a la lógica y restricciones del negocio. Las ventajas se resumen en la libertas y control que tiene el programador para interactuar con la base de datos al nivel de aislamiento que ésta le permita, con la lógica transaccional que él desee y utilizando los comandos e interfaces que él encuentre cómodos.

LO MALO

Contenedor de aplicaciones:

- Las transacciones que requieran de configuraciones concretas o manejen problemas transaccionales particulares no podrán ser realizadas correctamente, podrá perderse consistencia en la información o no cumplir con las reglas de negocio, en el peor de los casos puede que las transacciones no funcionen.
- El comportamiento transaccional es independiente de la lógica del negocio. Las operaciones son establecidas y están dadas por defecto.
- El contenedor maneja toda la interacción con la base de datos, incluyendo los niveles de aislamiento. Cambiar este comportamiento no es recomendable.

Programador:

- Se necesita un alto nivel de conocimiento y talento por parte del programador para que su configuración mantenga la consistencia e integridad de los datos y no ocasione fallas ni pérdida de datos.
- Es tedioso porque el programador comienza la transacción, la termina y la anula.
- La implementación de la transacción es más difícil, el programador debe adaptar los módulos a ambientes extraños uno por uno.

5. Bibliografía

Almenares, F., Bastente, P. (s.f.). *Transacciones y Seguridad en J2EE.* Universidad Carlos III de Madrid. Recuperado el: 20 de Octubre de 2013 de http://ocw.uc3m.es/ingenieria-telematica/software-de-comunicaciones-1/UDs_JEE/jee-unidad-4.

Database Administrator's Guide. (2008, March 13). Retrieved February 18, 2017, from https://docs.oracle.com/cd/B28359_01/server.111/b28310/create003.htm#ADMIN11073

Universidad De Granada (n.d.). Diseño Lógico: Diseño de bases de datos relacionases. Retrieved February 17, 2017, from http://elvex.ugr.es/idbis/db/docs/design/5-logical.pdf

Universidad De Buenos Aires (n.d.). Ingeniería de Software: Casos de Uso. Retrieved February 17, 2017, from http://www-2.dc.uba.ar/materias/isoft1/2001_2/apuntes/CasosDeUso.pdf