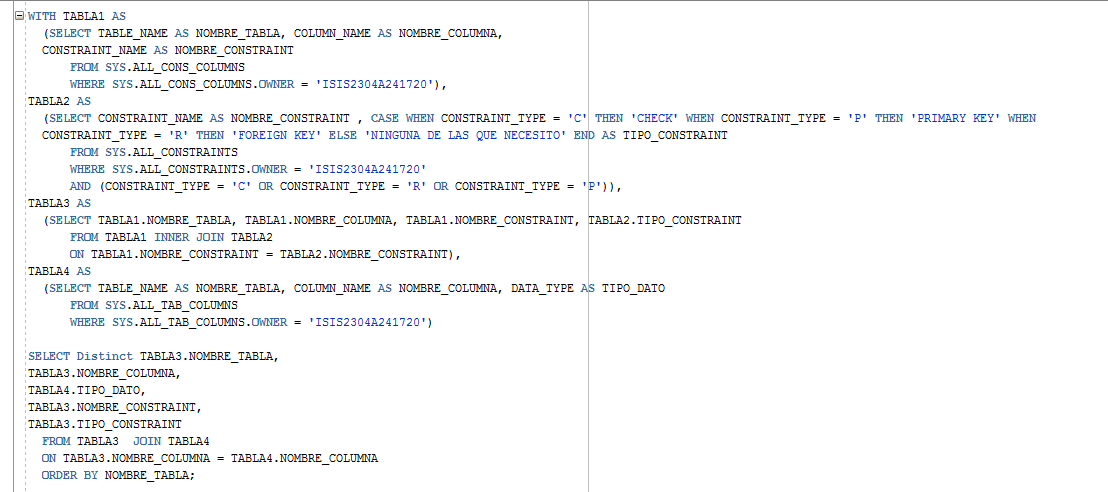
DOCUMENTO ITERACION 3

2.a. Diseño de la aplicación

Para introducir los nuevos requerimientos de la aplicación solo necesitamos crear dos atributos en la clase boleta, uno guarda el abonamiento al que pertenece y null si no pertenece a ningún abonamiento y el otro guarda un registro donde se sabe si la boleta sigue activa o fue devuelta. En cuanto a las tablas debimos incluir dos columnas en la tabla boleta donde se almacena esta información también y además para tener un registro de las notas debito que se han generado, se les asigna un ID único a cada una y en una tabla llamada devolución donde se guarda este ID, el ID de la boleta a la que esté relacionado sin ser foreign key y además el ID del cliente al que le perteneci esa boleta

El diseño de las tablas que le dan respaldo a la persistencia se basa en guardar la información disponible sobre el festival, los espectáculos, sus funciones, los sitios que se guardan en el sistema y toda la información relevante de estos. Adicional a esto manejamos una tabla donde se guardan las boletas que se vendieron que es donde más se manejan las transacciones del programa.



WITH TABLA1 AS

(SELECT TABLE\_NAME AS NOMBRE\_TABLA, COLUMN\_NAME AS NOMBRE\_COLUMNA,

CONSTRAINT\_NAME AS NOMBRE\_CONSTRAINT

FROM SYS.ALL\_CONS\_COLUMNS

WHERE SYS.ALL\_CONS\_COLUMNS.OWNER = 'ISIS2304A241720'),

TABLA2 AS

(SELECT CONSTRAINT\_NAME AS NOMBRE\_CONSTRAINT, CASE WHEN CONSTRAINT\_TYPE = 'C' THEN 'CHECK' WHEN CONSTRAINT\_TYPE = 'P' THEN 'PRIMARY KEY' WHEN

CONSTRAINT\_TYPE = 'R' THEN 'FOREIGN KEY' ELSE 'NINGUNA DE LAS QUE NECESITO' END AS TIPO\_CONSTRAINT

FROM SYS.ALL\_CONSTRAINTS

WHERE SYS.ALL\_CONSTRAINTS.OWNER = 'ISIS2304A241720'

AND (CONSTRAINT\_TYPE = 'C' OR CONSTRAINT\_TYPE = 'R' OR CONSTRAINT\_TYPE = 'P')),

TABLA3 AS

(SELECT TABLA1.NOMBRE\_TABLA, TABLA1.NOMBRE\_COLUMNA, TABLA1.NOMBRE\_CONSTRAINT, TABLA2.TIPO\_CONSTRAINT

FROM TABLA1 INNER JOIN TABLA2

ON TABLA1.NOMBRE\_CONSTRAINT = TABLA2.NOMBRE\_CONSTRAINT),

TABLA4 AS

(SELECT TABLE\_NAME AS NOMBRE\_TABLA, COLUMN\_NAME AS NOMBRE\_COLUMNA, DATA\_TYPE AS TIPO\_DATO

FROM SYS.ALL\_TAB\_COLUMNS

WHERE SYS.ALL\_TAB\_COLUMNS.OWNER = 'ISIS2304A241720')

SELECT Distinct TABLA3.NOMBRE\_TABLA,

TABLA3.NOMBRE\_COLUMNA,

TABLA4.TIPO\_DATO,

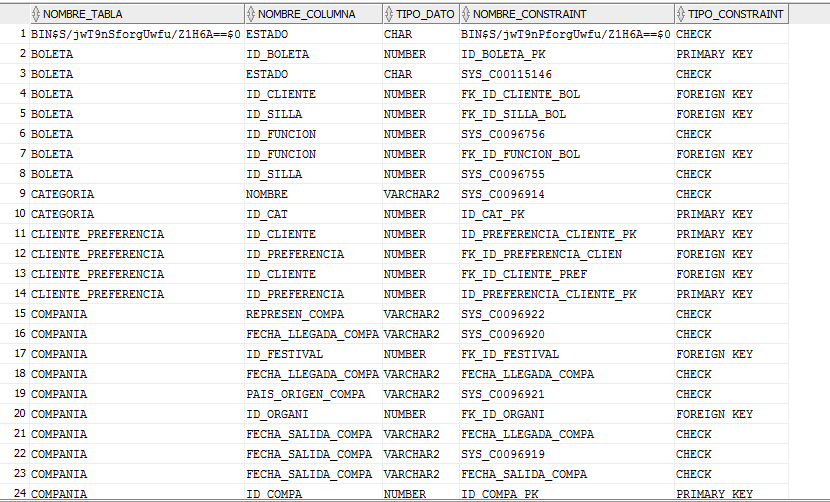
TABLA3.NOMBRE\_CONSTRAINT,

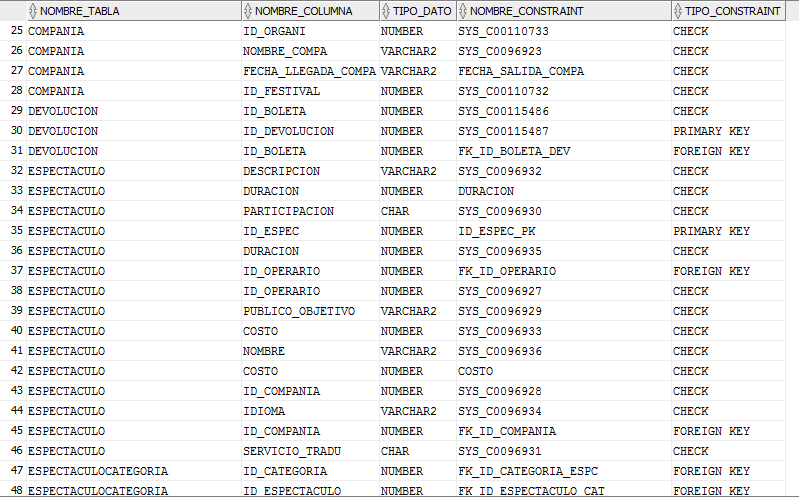
TABLA3.TIPO\_CONSTRAINT

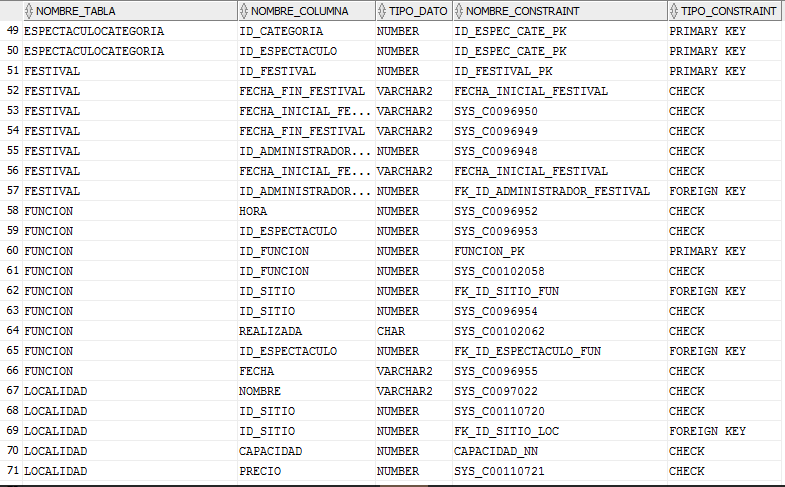
FROM TABLA3 JOIN TABLA4

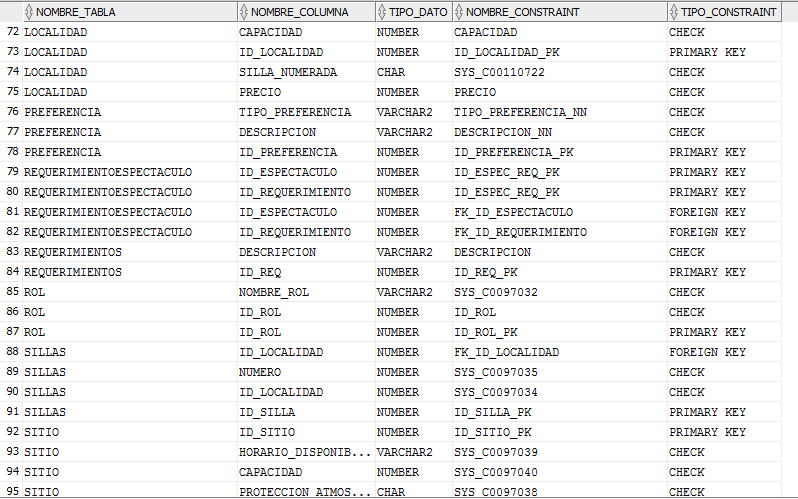
ON TABLA3.NOMBRE\_COLUMNA = TABLA4.NOMBRE\_COLUMNA

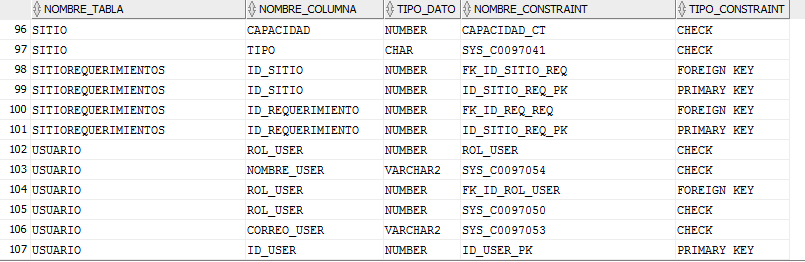
ORDER BY NOMBRE\_TABLA;











2.c. ACID

Para garantizar las propiedades ACID de los requerimientos usamos las siguientes estrategias, usaremos como ejemplo el método de crear abonamiento

ATOMICIDAD:

Para garantizar la atomicidad del requerimiento cuando comenzamos un método desactivamos el autocommit, y comenzamos a intentar crear el abonamiento y sus respectivas boletas.

COHERENCIA:

Para garantizar la coherencia, almacenamos si las boletas pertenecen o no a un abonamiento en una columna de esta tabla y cada boleta debe llevar este numero de abonamiento. Al no salirnos del método donde declaramos el numero del abonamiento garantizamos que todas las boletas se almacenen con el mismo numero.

AISLAMIENTO:

Para garantizar el aislamiento de la operación usamos el comando

conn.setTransactionIsolation(Connection.*TRANSACTION\_SERIALIZABLE*);

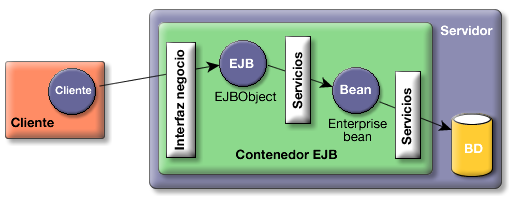
Este garantiza que esta sea la única transacción que se realiza en el momento

PERMANENCIA:

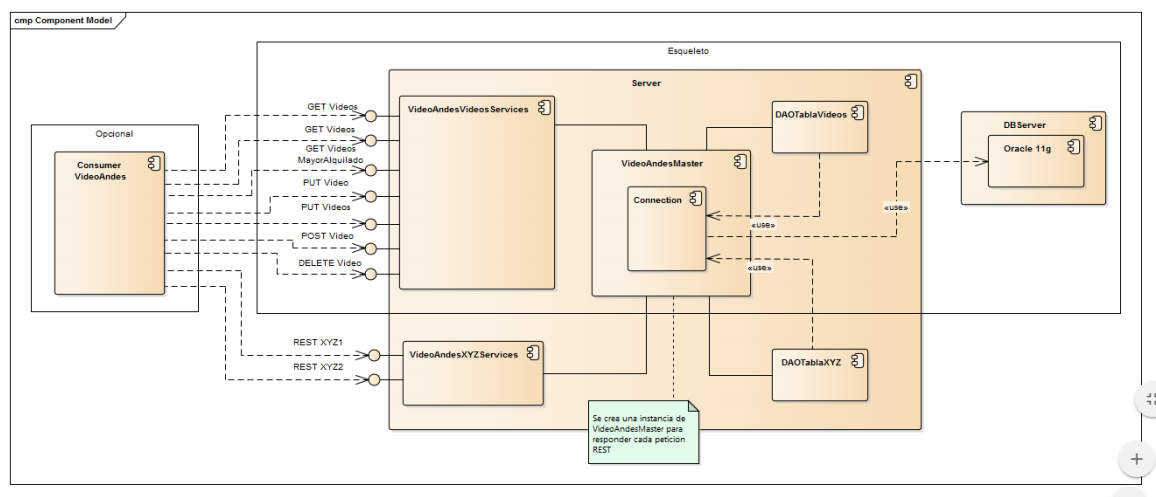
Apenas terminamos de realizar una operación con éxito, volvemos a activar el autocommit y además hacemos commit de las operaciones que acabamos de llevar a cabo para que los datos entren o salgan de la base de datos y los cambios se mantengan en el sistema.

Bono:

La principal diferencia entre el manejo transaccional entre un contenedor de aplicaciones como EJB y el programador de la aplicación es que estos contenedores se encargan de abrir y cerrar las transacciones cuando se llama un método del bean, mientras que por parte del programador de la aplicación este se debe encargar de abrirlas y cerrarlas correctamente para garantizar las propiedades ACID de la aplicación. Cuando se usa un contenedor la arquitectura del software es la siguiente:



El cliente se comunica con el contenedor y este con la base de datos, la transaccionalidad está a cargo del contenedor y este se encarga de mantener la coherencia en las bases de datos. Sin embargo, podemos ver diferencias ya que este se encarga de todo mientras que en nuestra arquitectura todo es muy detallado e independiente. Nuestra arquitectura es parecida a esta:



En esta podemos ver que tenemos una clase DAO para cada recurso que usamos y el cliente no se comunica directamente con esta clase. En estas clases es donde nos comunicamos con la base de datos y es en este punto donde el programador debe garantizar la transaccionalidad de los métodos. Además, todo el proceso está a cargo de el programador, como llega la instrucción hasta la base de datos y como le llega la respuesta al cliente.

Encontramos que el uso de los contenedores tiene una ventaja sobre el manejo por parte del programador ya que se garantiza la coherencia de los datos dentro de la base de datos y maneja todos los métodos como una transacción. Sin embargo, esto puede ser una desventaja si un método requiere el llamado de otros métodos y se debe verificar que todos se puedan ejecutar antes de que los datos entren a la base de datos. Un ejemplo de esto es vender múltiples boletas, existe un método que vende una boleta, y este hace commit al terminar de vender la boleta, sin embargo, cuando estamos vendiendo múltiples boletas necesitamos ejecutar el método varias veces y en este caso necesitamos que el commit se haga al final de la transacción, es decir al final de la venta de todas y que haya sido posible crear todas las boletas. En un contenedor de aplicaciones tocaría hacer el proceso en un solo método para que se garantice la transaccionalidad correctamente ya que si llamamos el método de vender boleta este insertaría cada boleta en la base de datos antes de verificar que todas se puedan insertar, y si alguna no se puede insertar las otras ya estarían dentro de la base de datos y no se podría hacer roll-back.