Análisis de operaciones SQL

Insertar compra correcta

Hibernate: select cliente0_.nif as nif1_0_0_, cliente0_.apellidos as apellidos2_0_0_, cliente0_.ciudad as ciudad3_0_0_, cliente0_.cp as cp4_0_0_, cliente0_.direccion as direccion5_0_0_, cliente0_.nombre as nombre6_0_0_ from HR.Cliente cliente0_ where cliente0_.nif=?

Hibernate: select grupo0_.idgrupo as idgrupo1_3_0_, grupo0_.activo as activo2_3_0_, grupo0_.estilo as estilo3_3_0_, grupo0_.nombre as nombre4_3_0_ from HR.Grupo grupo0_ where grupo0_.idgrupo=?

Hibernate: select concierto0_.idconcierto as idconcierto1_2_, concierto0_.ciudad as ciudad2_2_, concierto0_.fecha as fecha3_2_, concierto0_.IDGRUPO as idgrupo7_2_, concierto0_.nombre as nombre4_2_, concierto0_.precio as precio5_2_, concierto0_.tickets as tickets6_2_ from HR.Concierto concierto0_ where concierto0_.fecha=? and concierto0_.IDGRUPO=?

Hibernate: select compra0_.idcompra as idcompra1_1_, compra0_.NIF as nif3_1_, compra0_.IDCONCIERTO as idconcierto4_1_, compra0_.N_TICKETS as n_tickets2_1_ from HR.Compra compra0_ where compra0_.idcompra=(select max(compra1_.idcompra) from HR.Compra compra1_)

Hibernate: insert into HR.Compra (NIF, IDCONCIERTO, N_TICKETS, idcompra) values (?, ?, ?, ?)

Hibernate: update HR.Concierto set ciudad=?, fecha=?, IDGRUPO=?, nombre=?, precio=?, tickets=? where idconcierto=?

Se realizan 4 operaciones de selección, 1 de inserción y 1 de modificación.

Este es un caso de prueba correcto en el que se deben de ejecutar las 3 primeras selects para comprobar que, cliente, grupo y concierto existen y son correctos. Después se ejecuta otra select para obtener el último identificador de compra utilizado, que será el mayor de los id's que se tengan en la tabla. Esto último es necesario para poder insertar la compra con un identificador válido y correcto.

Una vez realizadas todas las comprobaciones se realiza la inserción de la compra.

Una vez insertada la compra se modifica la disponibilidad de tickets para el concierto.

Insertar compra con grupo incorrecto

Hibernate: select cliente0_.nif as nif1_0_0_, cliente0_.apellidos as apellidos2_0_0_, cliente0_.ciudad as ciudad3_0_0_, cliente0_.cp as cp4_0_0_, cliente0_.direccion as

direccion5_0_0_, cliente0_.nombre as nombre6_0_0_ from HR.Cliente cliente0_ where cliente0_.nif=?

Hibernate: select grupo0_.idgrupo as idgrupo1_3_0_, grupo0_.activo as activo2_3_0_, grupo0_.estilo as estilo3_3_0_, grupo0_.nombre as nombre4_3_0_ from HR.Grupo grupo0_ where grupo0_.idgrupo=?

Este es un caso de prueba incorrecto (se espera que la transacción no finalice y lance una excepción), donde se intenta insertar una compra con un grupo que no existe. En este caso se ejecutan dos operaciones de selección. La primera comprueba que existe el cliente y la segunda que existe el grupo. Al no existir lanza una excepción y cierra la transacción sin cometerla.

Se podría realizar de modo que solo ejecutara la comprobación del grupo, ahorrándonos comprobar el cliente, pero en el caso de prueba de cliente inexistente, forzosamente se debería de comprobar el grupo primero. Por lo que realmente no estaríamos ahorrándonos una operación. La operación que nos "ahorramos" en este caso de prueba se "pagaría" en próximos casos de prueba.

Para tratar de ahorrar, en la práctica, consultas a la base de datos, podríamos estudiar las probabilidades de que se realice la transacción con un cliente incorrecto o con un grupo incorrecto para, consultar primero el campo que más probabilidades tiene de ser incorrecto. Así, en promedio, ahorraremos consultas y mejorará la eficiencia de la aplicación.

Insertar compra con cliente incorrecto

Hibernate: select cliente0_.nif as nif1_0_0_, cliente0_.apellidos as apellidos2_0_0_, cliente0_.ciudad as ciudad3_0_0_, cliente0_.cp as cp4_0_0_, cliente0_.direccion as direccion5_0_0_, cliente0_.nombre as nombre6_0_0_ from HR.Cliente cliente0_ where cliente0_.nif=?

En este caso de prueba incorrecto, solo se realiza una operación de consulta a la base de datos, ya que es la primera comprobación de la transacción. Al no existir el cliente se lanza una excepción y no se realizan más consultas a lavase de datos.

Las observaciones dadas en el apartado anterior (insertar compra con grupo incorrecto), son también válidas para este caso de prueba. En este caso de prueba, nos estamos "ahorrando" la consulta que "pagamos" en el anterior.

Insertar compra con concierto incorrecto

Hibernate: select cliente0_.nif as nif1_0_0_, cliente0_.apellidos as apellidos2_0_0_, cliente0_.ciudad as ciudad3_0_0_, cliente0_.cp as cp4_0_0_, cliente0_.direccion as direccion5_0_0_, cliente0_.nombre as nombre6_0_0_ from HR.Cliente cliente0_ where cliente0_.nif=?

Hibernate: select grupo0_.idgrupo as idgrupo1_3_0_, grupo0_.activo as activo2_3_0_, grupo0_.estilo as estilo3_3_0_, grupo0_.nombre as nombre4_3_0_ from HR.Grupo grupo0_ where grupo0_.idgrupo=?

Hibernate: select concierto0_.idconcierto as idconcierto1_2_, concierto0_.ciudad as ciudad2_2_, concierto0_.fecha as fecha3_2_, concierto0_.IDGRUPO as idgrupo7_2_, concierto0_.nombre as nombre4_2_, concierto0_.precio as precio5_2_, concierto0_.tickets as tickets6_2_ from HR.Concierto concierto0_ where concierto0_.fecha=? and concierto0_.IDGRUPO=?

En este caso de prueba se realizan 3 operaciones de consulta para comprobar que existan cliente, grupo y concierto. Como las comprobaciones deben realizarse unas antes que otras, en alguno de estos casos de prueba se realizarán todas las comprobaciones hasta ver que el último campo que se comprueba es inválido. Como ya se ha comentado, lo mejor en estos casos es realizar las comprobaciones desde el campo con más posibilidades o que más veces se introduce de forma errónea hasta el que tenga menor probabilidad o se introduzca menos veces de forma errónea, teniendo en cuenta que la comprobación del concierto debe hacerse obligatoriamente después de comprobar el grupo, ya que se necesita conocer el grupo para consultar el concierto por fecha y grupo.

Insertar compra sin tickets con concierto incorrecto

Hibernate: select cliente0_.nif as nif1_0_0_, cliente0_.apellidos as apellidos2_0_0_, cliente0_.ciudad as ciudad3_0_0_, cliente0_.cp as cp4_0_0_, cliente0_.direccion as direccion5_0_0_, cliente0_.nombre as nombre6_0_0_ from HR.Cliente cliente0_ where cliente0_.nif=?

Hibernate: select grupo0_.idgrupo as idgrupo1_3_0_, grupo0_.activo as activo2_3_0_, grupo0_.estilo as estilo3_3_0_, grupo0_.nombre as nombre4_3_0_ from HR.Grupo grupo0_ where grupo0_.idgrupo=?

Hibernate: select concierto0_.idconcierto as idconcierto1_2_, concierto0_.ciudad as ciudad2_2_, concierto0_.fecha as fecha3_2_, concierto0_.IDGRUPO as idgrupo7_2_, concierto0_.nombre as nombre4_2_, concierto0_.precio as precio5_2_, concierto0_.tickets as tickets6_2_ from HR.Concierto concierto0_ where concierto0_.fecha=? and concierto0_.IDGRUPO=?

Se realizan 3 operaciones de consulta, ya que la transacción primero comprueba que cliente, grupo y concierto existen. Se podría argumentar algo similar a los apartados previos teniendo en cuenta que podría comprobarse el número de tickets disponibles antes que el cliente, pero obligatoriamente debe comprobarse después de grupo y concierto, ya que necesitamos conocer el concierto para el que se solicitan tickets y, se necesita conocer el grupo para acceder a la información del concierto.

Información completa con grafos de entidades

Hibernate: select grupo0_.idgrupo as idgrupo1_3_0_, conciertos1_.idconcierto as idconcierto1_2_1_, compras2_.idcompra as idcompra1_1_2_, cliente3_.nif as nif1_0_3_, grupo0_.activo as activo2_3_0_, grupo0_.estilo as estilo3_3_0_, grupo0_.nombre as nombre4_3_0_, conciertos1_.ciudad as ciudad2_2_1_, conciertos1_.fecha as fecha3_2_1_, conciertos1_.IDGRUPO as idgrupo7_2_1_, conciertos1_.nombre as nombre4_2_1_, conciertos1_.precio as precio5_2_1_, conciertos1_.tickets as tickets6_2_1_, conciertos1_.IDGRUPO as idgrupo7_2_0__, conciertos1_.idconcierto as idconcierto1_2_0__, compras2_.NIF as nif3_1_2_, compras2_.IDCONCIERTO as idconcierto4_1_2_, compras2_.N_TICKETS as n_tickets2_1_2_, compras2_.IDCONCIERTO as idconcierto4_1_1__, compras2_.idcompra as idcompra1_1_1__, cliente3_.apellidos as apellidos2_0_3_, cliente3_.ciudad as ciudad3_0_3_, cliente3_.cp as cp4_0_3_, cliente3_.direccion as direccion5_0_3_, cliente3_.nombre as nombre6_0_3_ from HR.Grupo grupo0_ left outer join HR.Concierto conciertos1_ on grupo0_.idgrupo=conciertos1_.IDGRUPO left outer join HR.Compra compras2_ on conciertos1_.idconcierto=compras2_.IDCONCIERTO left outer join HR.Cliente cliente3_ on compras2_.NIF=cliente3_.nif

Este caso prueba realiza una única operación SQL, ya que estamos obteniendo toda la información de una sola vez mediante un grafo de entidades y dos subgrafos, por lo que hibernate genera una única operación.

Si realizamos esto sin usar un grafo y subgrafos se daría el problema de n+1 consultas. En este caso, se deberían de haber realizado una consulta por grupo más una consulta por cada concierto asociado. A su vez hay que sumar que por cada concierto se deben realizar tantas consultas como compras asociadas al mismo se tengan en la base de datos multiplicado por 2, ya que no solo se debe consultar la compra, sino que se debería consultar además por cada compra el cliente asociado a esta.

Si se tienen m grupos con una media de n conciertos por grupo, que a su vez tienen una media de k compras, se tiene que en promedio se realizarían 2k consultas por cada uno de los nm conciertos totales, lo que hace que en promedio se realicen 2nmk consultas.

En este caso concreto, en la base de datos se tienen 2 grupos, uno con un concierto y otro con dos conciertos. De los tres conciertos, uno tiene 4 compras (las 3 iniciales más la que se inserta en un caso de prueba anterior) y los otros dos 1 compra.

Sin usar grafos se tendrían 2 consultas para los grupos más 3 consultas para los conciertos, más 2·6 consultas para las compras y clientes. En total se hubiesen realizado 17 consultas a la base de datos frente a una única consulta.

Desactivar grupo correcto

Hibernate: select grupo0_.idgrupo as idgrupo1_3_0_, grupo0_.activo as activo2_3_0_, grupo0_.estilo as estilo3_3_0_, grupo0_.nombre as nombre4_3_0_ from HR.Grupo grupo0_ where grupo0_.idgrupo=?

Hibernate: select concierto0_.idconcierto as idconcierto1_2_, concierto0_.ciudad as ciudad2_2_, concierto0_.fecha as fecha3_2_, concierto0_.IDGRUPO as idgrupo7_2_, concierto0_.nombre as nombre4_2_, concierto0_.precio as precio5_2_, concierto0_.tickets as tickets6_2_ from HR.Concierto concierto0_ where concierto0_.IDGRUPO=?

Hibernate: delete from HR.Compra where IDCONCIERTO=?

Hibernate: update HR.Grupo set activo=?, estilo=?, nombre=? where idgrupo=?

Hibernate: delete from HR.Concierto where idconcierto=?

En un principio esta transacción estaba implementada usando el método remove que heredaba la clase DAO de JpaDAO. En ese caso se estaban realizando 13 operaciones SQL que, aunque en principio no parecen muchas, si aplicamos esto a una basede datos de dimensiones reales, la cantidad de consultas podría ser muy grande. Gracias al debbuger de hibernate se pudo identificar esto. Se reviso y optimizó la transacción usando un bulk delete para eliminar todas las compras asociadas a un mismo concierto, reduciendo considerablemente el número de operaciones, de 13 a 5. La operación de bulk delete se realiza directamente contra la base de datos y hay que tener precaución al usarla, pero en este caso creo que está justificado su uso. Tal y como está implementada la transacción, no debería de dar problemas el uso de esta forma de eliminar datos de la base de datos.

La primera operación se genera al buscar el grupo para comprobar que existe.

La segunda operación busca todos los conciertos que solicitamos en la transacción.

La tercera elimina todas las compras asociadas al concierto consultado en la operación anterior.

La cuarta es la que se genera al modificar el estado del grupo a 0.

La última transacción elimina el concierto.

Si nos fijamos en el fichero script.sql y en el test implementado vemos que se desactiva el grupo 1, el cual solo tiene un concierto.

Desactivar grupos borra los registros asociados

En este caso de prueba nos aprovechamos del caso de prueba anterior y miramos si se borraron todos los registros correctamente sin realizar ninguna operación SQL. Si hubiésemos repetido el proceso de desactivar un grupo activo, se deberían de haber ejecutado varias operaciones SQL, según el número de conciertos asociados al grupo y el número de compras asociados a cada concierto.

Desactivar grupo incorrecto

Hibernate: select grupo0_.idgrupo as idgrupo1_3_0_, grupo0_.activo as activo2_3_0_, grupo0_.estilo as estilo3_3_0_, grupo0_.nombre as nombre4_3_0_ from HR.Grupo grupo0_ where grupo0_.idgrupo=?

Solo se realiza una única operación de consulta, ya que lo primero que hace la transacción correspondiente es comprobar que el grupo indicado existe. Como en este caso no existe, solo se realiza la operación de consulta del grupo y lanza una excepción.