**Práctica 3 Semana 6 y 7**

**APARTADO A**

**USUARIOS:** Interactúan con la base de datos de diferentes formas según lo que necesiten.

**GESTIÓN DE ESPACIO(TABLESPACE):** Es la ubicación de almacenamiento donde se guardan los datos de los objetos.

**TABLAS PÚBLICAS:** Tabla a la que se puede acceder.

**TRIGGERS:** Se ejecuta cuando tiene lugar un evento sobre las tablas.

**PROCEDIMIENTOS:** Es un bloque de código que se puede invocar desde cualquier punto del programa.

**A.0.**

Entramos como Admin User, creamos el espacio de tablas y al crear el usuario le asignamos los permisos básicos (CREATE, DELETE TABLE…), también para uso de procedimientos y disparadores.

**A.1.**

Creamos la tabla (CREATE TABLE NOTIFICACIONES).

**A.2.**

Ponemos la tabla pública.

**A.4. (ANTES QUE A.3.)**

Las explicaciones de su funcionamiento se encuentran en el código.

**A.3.**

Se trata de la estructura de un trigger en la cual se llama al procedimiento “regalacomisiones” pasándole por parámetro los valores.

**APARTADO F (ANTES QUE AL APARTADO B)**

Hay muchas transacciones por que son todas transacciones autónomas (hebras independientes a la principal) todas tienen su propio identificador.

Transaction ISOLATION correcto.

LOCAL\_TRANSACTION\_ID

1.5.1229

Sequence SEC\_TRANS\_1 borrado.

Sequence SEC\_TRANS\_1 creado.

Confirmación terminada.

Trans. Principal Empieza: 2.3.1460

se ha dormido -> antes: 0 después: 0 Num.Trans.Secun: 10.30.1677

se ha dormido -> antes: 1 después: 1 Num.Trans.Secun: 10.28.1677

se ha dormido -> antes: 0 después: 0 Num.Trans.Secun: 7.18.1218

se ha dormido -> antes: 1 después: 1 Num.Trans.Secun: 8.24.1218

se ha dormido -> antes: 0 después: 0 Num.Trans.Secun: 5.22.1247

se ha dormido -> antes: 1 después: 1 Num.Trans.Secun: 5.26.1248

se ha dormido -> antes: 0 después: 0 Num.Trans.Secun: 9.3.1264

se ha dormido -> antes: 1 después: 1 Num.Trans.Secun: 4.23.1202

Trans. Principal TERMINA: 2.3.1460

**APARTADO B**

**CHECKPOINTS:** Es el punto de guardado de la transacción.

**ROLLBACK:** Elimina en caso de error todo el procedimiento que haya realizado la transacción. Si hubiera un checkpoint vuelve hasta él.

**TRANSACCIÓN AUTÓNOMA:** Es una transacción que se puede llamar desde otra.

Aplicamos una transacción autónoma para el correcto funcionamiento.

En el caso de que la suma total supere los 100 euros, hacemos rollback. esto elimina las modificaciones que haya podido hacer la transacción a partir del check point.

Al eliminarlo, vuelve a la línea del check point, donde sigue el proceso con los datos cargados antes de este.

**APARTADO C**

Al igual que el procedure, se encuentra explicado en el código.

**APARTADO D**

**BLOQUEO EXPLÍCITO:** Previene que otro usuario seleccione datos de la tabla o cargue datos en ella.

Después de comprobar que el número aleatorio coincide bloqueamos la tabla para que nadie pueda modificarla mientras estamos insertando(utilizando ‘LOCK TABLE’).

**APARTADO E**

Unas líneas más abajo de bloquear la tabla comprobamos si la suma de comisiones es mayor que 1000 hacemos rollback y por lo tanto deshacemos el insert y se desbloquea la tabla.

En caso de que la suma no sea mayor que 1000 hacemos un commit se guardan los cambios y se desbloquea la tabla.

**APARTADO H**

Cuando en los resultados deba predominar la fiabilidad respecto de la rapidez. En este caso evitaremos lecturas fantasmas.

**APARTADO I**

Cuando queremos que predomine la rapidez a la fiabilidad de los resultados. En este caso evitaremos lecturas sucias pero permitiremos lecturas no repetibles.

**APARTADO J**

**j.1.** Usaría el nivel de aislamiento Read Commited ya que las transacciones de lectura solo ven datos confirmados antes de empezar y no permiten modificaciones de los mismos. Por tanto en este caso priorizamos que los datos sean rápidos a que sean seguros. Usaría la instrucción SET TRANSACTION ISOLATION LEVEL READ COMMITTED; en la transacción de actualización de los datos y haría commit cada vez que estos datos cambiarán para priorizar que en todo momento esos datos desde otras transacciones se vean cuanto antes

**j.2.** Serializable,ya que priorizaría la fiabilidad (habría que esperar a que la transacción termine) para asegurarse de que la puerta está cerrada y no se produzca una lectura no deseada.Las transacciones son cortas.

**j.3.** Read Commited, ya que necesitamos que los datos tengas cierta integridad y estén confirmados (sin rollback)

**j.4.** Una transacción de actualización y serializable ya que se trata de una transacción corta, no pierde la actualización, se garantizan lecturas repetibles.

**APARTADO K**

**k.1**: si se ejecuta el primer rollback que va a al savepoint s1, la siguiente instrucción que ejecutará será el siguiente if (el segundo). Si entra en él, saltará una excepción que indicará que el savepoint s2 no ha sido nunca creado. Esto es porque al hacer el rollback se “deshace” la instrucción del savepoint.

**k.2**: cuando ejecutamos el primer rollback este “deshará” todos los cambios hasta la última confirmación en el procedimiento. La siguiente instrucción que se ejecutará será el rollback to savepoint s1. Este savepoint no esta definido porque el rollback anterior lo ha borrado.

Alejandro Ruiz Martín

Francisco Rafael García Rofes

Héctor Arranz Torres

Sandra Alonso Paz

Verónica Calzada Álvarez

Ampliación de Bases de Datos 3ºE