MBDA Grupo 3/ 301

AUTOESTUDIO #5

Nombres:

1. Martínez Núñez Gerónimo
2. Silva Rodríguez Sergio Andrey

INVESTIGACION TRANSACCIONES

A. TRANSACCIONES

1. ¿Cómo se define el comienzo y fin de una transacción en ORACLE?

En Oracle, el inicio y el final de una transacción se marcan mediante el uso de comandos específicos. Para comenzar una transacción, simplemente ejecutas "BEGIN TRANSACTION" o simplemente empiezas a realizar las operaciones que deseas incluir en la transacción.

Ejemplo:

```
BEGIN
    -- Realizar operaciones de la transacción aquí
    UPDATE empleados SET salario = salario * 1.1 WHERE departamento_id = 10;
END;
```

Una vez que hayas completado las acciones deseadas y estés listo para confirmar los cambios, utilizas el comando "COMMIT" para finalizar la transacción y hacer que los cambios sean permanentes en la base de datos.

Ejemplo:

```
COMMIT;
```

Si en algún momento decides revertir los cambios y deshacer las operaciones realizadas durante la transacción, puedes utilizar el comando "ROLLBACK".

Ejemplo:

```
ROLLBACK;
```

Estos comandos son como el punto de inicio y el punto final de una historia en la base de datos de Oracle: empiezas a escribir tu historia con BEGIN TRANSACTION, la finalizas y la guardas con COMMIT, o la descartas y la borras con ROLLBACK.

2. ¿Cuáles son los diferentes tipos de aislamiento que soporta ORACLE? Para cada uno de ellos detalle, ¿cómo maneja los bloqueos? ¿qué problemas resuelve?

Oracle ofrece varios niveles de aislamiento para gestionar transacciones, y cada uno aborda de manera distinta la concurrencia y la consistencia de la base de datos. El nivel de aislamiento más bajo es "READ COMMITTED", que permite a una transacción ver solo los cambios confirmados por otras transacciones. Este nivel evita la lectura de datos no confirmados, pero aún puede provocar problemas de lecturas inconsistentes.

Ejemplo:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
-- Operaciones de la transacción
COMMIT;
```

El siguiente nivel es "SERIALIZABLE", que impide cualquier lectura o escritura concurrente, asegurando una mayor consistencia, pero a costa de una mayor contención y posiblemente reduciendo el rendimiento en entornos de alta concurrencia.

Ejemplo:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
-- Operaciones de la transacción
COMMIT;
```

Oracle también implementa mecanismos de bloqueo, como bloqueo de filas o bloqueo de tablas, para gestionar la concurrencia. El bloqueo de filas permite que varias transacciones trabajen en diferentes conjuntos de datos simultáneamente, pero puede llevar a bloqueos y espera, afectando el rendimiento.

Ejemplo:

```
-- Dentro de una transacción
SELECT * FROM empleados WHERE departamento_id = 10 FOR UPDATE;
-- Operaciones de actualización o inserción
COMMIT;
```

En contraste, el bloqueo de tablas bloquea toda la tabla durante una transacción, garantizando consistencia, pero limitando la concurrencia.

Ejemplo:

```
-- Dentro de una transacción
LOCK TABLE empleados IN EXCLUSIVE MODE;
-- Operaciones de actualización o inserción
COMMIT;
```

3. ¿Cuál es el tipo de aislamiento por defecto en ORACLE?

En Oracle, el tipo de aislamiento por defecto es "READ COMMITTED". Esto significa que, de manera predeterminada, una transacción en Oracle puede ver solo los cambios confirmados por otras transacciones.

B. VISTAS

1. ¿Cuáles son los mecanismos para la creación y borrado de vistas en ORACLE?

En Oracle, crear y borrar vistas implica un conjunto sencillo de comandos SQL. Para crear una vista, primero defines la consulta que quieres encapsular y luego ejecutas el comando CREATE VIEW.

Ejemplo:

```
CREATE VIEW nombre_vista AS
SELECT columna1, columna2
FROM nombre_tabla
WHERE condicion;
```

Para borrar una vista, utilizas el comando DROP VIEW.

Ejemplo:

```
DROP VIEW nombre_vista;
```

2. ¿Cuáles son las restricciones de las vistas en ORACLE?

En Oracle, las vistas ofrecen una forma práctica de organizar y presentar datos, pero tienen algunas restricciones a considerar. En primer lugar, una vista no puede contener órdenes de manipulación de datos **(INSERT, UPDATE, DELETE)** a menos que se establezcan condiciones específicas, como reglas de integridad. Además, las vistas **no pueden referenciar o contener operaciones sobre otras vistas** que estén basadas en consultas que involucren uniones externas, subconsultas correlacionadas o funciones de usuario definidas en SQL. También es importante señalar que las columnas en la vista deben tener nombres únicos, y no se pueden utilizar cláusulas **ORDER BY** en la definición de la vista, ya que el orden de los resultados puede gestionarse al consultar la vista en lugar de definirse directamente en ella.

C. MODULARIDAD PAQUETES

1. ¿Para qué sirve un paquete?

Un paquete en Oracle es como un conjunto organizado de procedimientos, funciones y variables que trabajan juntos para realizar una tarea específica. Es como un paquete de herramientas que puedes utilizar en tu base de datos.

2. ¿Cuáles son los mecanismos para la creación, invocación, modificación y borrado de paquetes en ORACLE?

En Oracle, la creación de paquetes implica definir un conjunto de procedimientos, funciones y variables que trabajan juntos y luego empaquetarlos en una unidad lógica.

Para crear un paquete, escribirías el código del paquete con todos los procedimientos y funciones necesarios y luego ejecutarías el comando CREATE PACKAGE para almacenarlo en la base de datos.

Ejemplo:

```
CREATE OR REPLACE PACKAGE MiPaquete AS
    PROCEDURE Saludar(p_nombre IN VARCHAR2);
    FUNCTION Duplicar(numero IN NUMBER) RETURN NUMBER;
END MiPaquete;
/
CREATE OR REPLACE PACKAGE BODY MiPaquete AS
    PROCEDURE Saludar(p_nombre IN VARCHAR2) IS
    BEGIN
        DBMS_OUTPUT.PUT_LINE('¡Hola, ' || p_nombre || '!');
    END Saludar;

    FUNCTION Duplicar(numero IN NUMBER) RETURN NUMBER IS
    BEGIN
        RETURN numero * 2;
    END Duplicar;
END MiPaquete;
/
```

La invocación de un paquete implica llamar a sus procedimientos o funciones utilizando la sintaxis adecuada, como si estuvieras utilizando cualquier otra función o procedimiento en tu código.

Ejemplo1: *Invocación de un procedimiento del paquete*

```
BEGIN
    MiPaquete.Saludar('Juan');
END;
/
```

Ejemplo2: *Invocación de una función del paquete*

```
DECLARE
    resultado NUMBER;
BEGIN
    resultado := MiPaquete.Duplicar(5);
    DBMS_OUTPUT.PUT_LINE('El resultado es: ' || resultado);
END;
/
```

Si necesitas realizar modificaciones en un paquete existente, puedes utilizar el comando ALTER PACKAGE para hacer ajustes en su código.

Ejemplo:

```
ALTER PACKAGE MiPaquete ADD PROCEDURE Despedirse(p_nombre IN VARCHAR2);
/
CREATE OR REPLACE PACKAGE BODY MiPaquete AS
    -- ... (código existente)

    PROCEDURE Despedirse(p_nombre IN VARCHAR2) IS
    BEGIN
        DBMS_OUTPUT.PUT_LINE('¡Adiós, ' || p_nombre || '!');
    END Despedirse;
END MiPaquete;
/
```

Finalmente, para borrar un paquete, puedes utilizar el comando DROP PACKAGE.

Ejemplo:

```
DROP PACKAGE MiPaquete;
```

D. SYS_REFCURSOR

1. ¿Qué es un SYS_REFCURSOR? ¿Para qué sirve?

Un `SYS_REFCURSOR` es esencialmente un tipo de dato especial en Oracle que actúa como un cursor o puntero a un conjunto de resultados en una consulta.

Este tipo de dato se utiliza comúnmente en el contexto de procedimientos almacenados o funciones para devolver conjuntos de resultados dinámicos. Sirve como una especie de intermediario que permite a los programas externos, como aplicaciones o incluso otros procedimientos almacenados, trabajar con conjuntos de datos de manera más flexible.

Al devolver un `SYS_REFCURSOR`, puedes proporcionar a los usuarios o a otras partes de tu código acceso a los resultados de una consulta sin necesidad de conocer la estructura exacta de las columnas de antemano.

2. ¿Cómo se define, se asigna y se retorna?

Definir, asignar y retornar un SYS_REFCURSOR en Oracle generalmente se realiza en el contexto de procedimientos almacenados o funciones.

Ejemplo:

```
CREATE OR REPLACE PROCEDURE obtener_empleados(p_cursor OUT SYS_REFCURSOR) AS
BEGIN
    OPEN p_cursor FOR
        SELECT * FROM empleados;
END obtener_empleados;
/
```

Asignar un SYS_REFCURSOR.

Ejemplo:

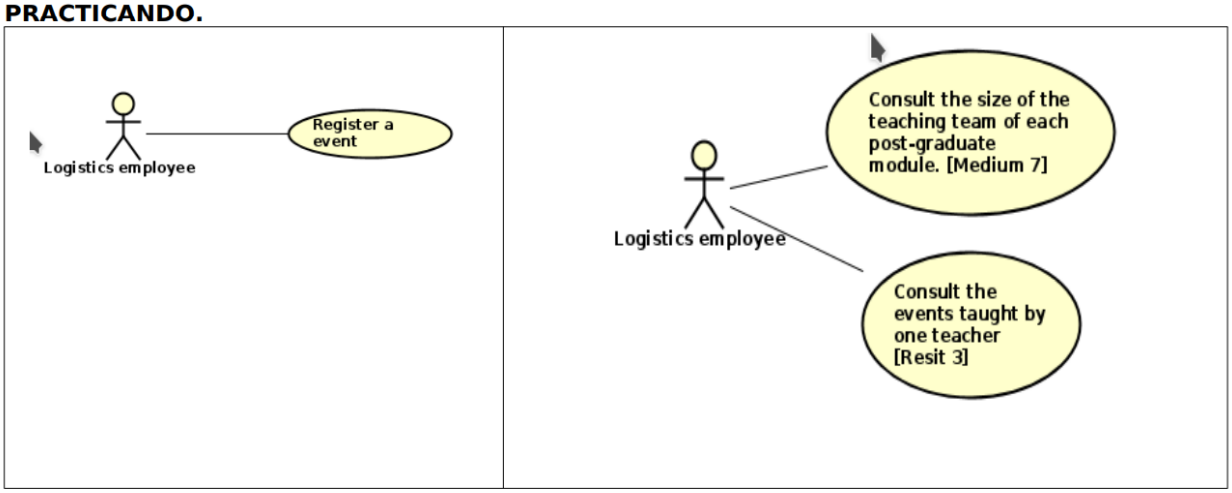
```
DECLARE
    mi_cursor SYS_REFCURSOR;
BEGIN
    obtener_empleados(mi_cursor);
    -- Puedes trabajar con el cursor mi_cursor aquí
END;
/
```

Retornar un SYS_REFCURSOR desde una función.

Ejemplo:

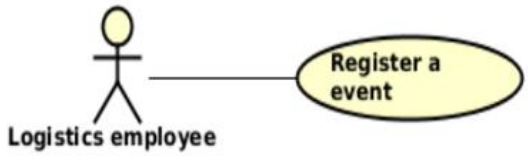
```
CREATE OR REPLACE FUNCTION obtener_empleados_func RETURN SYS_REFCURSOR IS
    mi_cursor SYS_REFCURSOR;
BEGIN
    OPEN mi_cursor FOR
        SELECT * FROM empleados;
    RETURN mi_cursor;
END obtener_empleados_func;
/
```

PRACTICANDO.



NOTAS

- El caso de uso Register an event está definido en el autoestudio 4.
- La primera consulta corresponde a medium 7
- La segunda consulta corresponde a una versión flexible de resit 3
- Las consultas retornan un CURSOR
  (ayuda: SYS_REFCURSOR Ver moodle)



Register an Event(Ad,Mo,Co,El)

Ad

El id se genera automáticamente: id del módulo, un punto, el tipo del evento y dos digitos aleatorio.
Si está programado a las 20:00 la duración debe ser una hora.

Mo

El único dato a modificar es el salón.
Se pueden adicionar profesores; pero no modificar ni eliminar.

Co

Consulta todos los datos del evento, incluyento sus profesores.

El

Siempre se puede eliminar

A. Ofreciendo servicios

1. Implemente los paquetes de componentes necesario para ofrecer las funciones básicas y consultas del ciclo actual del sistema (CRUD).

PC_EV [Consultar diseño al final]

(CRUDE (la especificación) , CRUDI (la implementación))

2. Proponga un caso de prueba exitoso por subprograma. (son seis)

(CRUDOK)

3. Proponga tres casos en los que el subprograma no se puede ejecutar.

(CRUDNoOK)

4. Escriba las instrucciones necesarias para eliminar los paquetes.

(CRUDX)

NOTA: Código adjuntado en la entrega - ***codigo.sql***