1. Uso de Github para coordinar el trabajo

El presente trabajo se ha coordinado a través de git y github, repartiendo la carga de trabajo y haciendo uso de las distintas versiones y ramas en todo momento. El repositorio, a través del cual nos hemos coordinado, se encuentra disponible en el siguiente enlace:

https://github.com/fjzamora93/AG4_BBDD_AVANZADAS

El github de los miembros del equipo se encuentra aquí:

https://github.com/fjzamora93/

https://github.com/alvarovr712/

JoseUrbano21 · GitHub

2. Preparativos

Para simplificar el trabajo, se decidió realizar la tarea a través de SQL Live de Oracle.

A modo de preparativo, se procedió a realizar las sentencias necesarias en SQL para crear la base de datos dispuesta en el ejercicio. Esto es, las 7 tablas propuestas. Dichas sentencias están guardadas en ficheros '.txt' de tal forma que sean accesibles en cualquier momento que se quiere utilizar SQL Live.

3. Creación de ramas y resolución individual

Cada miembro del equipo ha creado su propia rama 'git checkout –b 'Nombre'' a través de la cual ha tratado de resolver el ejercicio de forma individual antes de consultar y comparar los resultados con el resto de miembros.

Haciendo esto, hemos conseguido que todos los miembros realicen exactamente el mismo esfuerzo y sean capaces de llevar a cabo las acciones resolutivas de forma independiente, pero ofreciendo soporte los unos a los otros en caso de que necesitásemos ayuda.

Una vez cada uno logró resolver el ejercicio de forma individual (bien o mal), este se puso en común (todavía sin realizar el merge) y se trató de llegar a un acuerdo sobre cuál era la forma correcta de resolver cada uno de los 4 puntos del ejercicio.

Tras la puesta en común, se realizó el merge a la rama principal.

4. Funcionamiento de las sentencias

Primer punto:

Creamos el bloque nombrado o procedimiento, insertándole un IF llamando a la función y así comprobar si el oficio existe para poder modificarlo. En caso de no existir la secuencia seguira por el ELSE e imprimira por consola el mensaje : 'El oficio no existe'.

```
CREATE OR REPLACE PROCEDURE cambiar_empleado(
    p_empleado IN OUT employees.employee_id%TYPE,
    p_oficio IN jobs.job_id%TYPE
) AS
    --Inicializamos una transacción autónoma en el Procedure que no se verá afectada por el bucle principal.
    PRAGMA AUTONOMOUS_TRANSACTION;

    --Creamos una var. local para insertar la fila seleccionada con el filtro de los parámetros de nuestro Procedimiento.
    empleado_completo employees%ROWTYPE;
BEGIN
    SELECT *
    INTO empleado_completo
    FROM employees
    WHERE employee_id = p_empleado;

    -- Llamamos a la función. Si devuelve TRUE, actualizará la tabla empleados.
    IF oficio_existe(p_oficio) THEN
        DBMS_OUTPUT.PUT_LINE('El empleado ' || empleado_completo.first_name || ' antes era: ' ||empleado_completo.job_id);
        empleado_completo.job_id := p_oficio;
        UPDATE employees
        SET job_id = empleado_completo.job_id
        WHERE employee_id = p_empleado;
        DBMS_OUTPUT.PUT_LINE('El empleado ' || empleado_completo.first_name || ' ahora es: ' || empleado_completo.job_id);

        --Si la condición se cumple con éxito, nuestra transacción autónoma hará el commit.
        COMMIT;

    --Si la condición NO se cumple con éxito, nuestra transacción hará un rollback.
    ELSE
        DBMS_OUTPUT.PUT_LINE('El oficio no existe');
        ROLLBACK;
    END IF;
END;
/
```

Segundo punto:

El segundo apartado es el primero que hemos hecho se trata de crear la función para poder llamarlo posteriormente en el primer punto que es el bloque nombrado (procedimiento).

```sql
CREATE OR REPLACE FUNCTION oficio_existe(
    trabajo jobs.job_id%TYPE
) RETURN BOOLEAN IS
    --Creamos nuestras propias variables locales para operar dentro de la función.
    v_job_id jobs.job_id%TYPE;
BEGIN
    SELECT job_id INTO v_job_id
    FROM jobs
    WHERE trabajo = jobs.job_id;

    RETURN TRUE;

--Puesto que nuestra función siempre va a devolver TRUE. Necesitamos levantar nuestra propia excepción.
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END oficio_existe;

```

Tercer punto:

En el tercer punto se nos pide crear un bloque anónimo dentro del que usemos el procedimiento creado anteriormente, que a su vez llama a la función también creada con anterioridad. A continuación vemos su correcta ejecución sin problemas:

```sql

DECLARE
    v_empleado_id employees.employee_id%TYPE;
    v_oficio jobs.job_id%TYPE;

BEGIN
--AVISO: LIVE SQL NO ADMITE INTRODUCIR POR TECLADO LOS DATOS. Dejamos comentada la línea correspondiente con el '&'.
    v_empleado_id := 100;
    v_oficio := 'AD_VP';
    --cambiar_empleado(&v_empleado_id, &v_oficio);
    cambiar_empleado(v_empleado_id, v_oficio);
    COMMIT;

-- Levantamos las excepciones
EXCEPTION
    WHEN OTHERS THEN
        -- Manejo de excepciones aquí
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
        ROLLBACK; -- En caso de error, realiza un rollback para deshacer los cambios
END;
/
```

Para verificar que efectivamente el cambio se ha comiteado correctamente, realizaremos un SELECT a la tabla de employees. Aquí podemos observar que efectivamente el cambio se ha aplicado.

```sql
SELECT employee_id, first_name, job_id FROM employees
WHERE employee_id = 100;
```

EMPLOYEE_ID	FIRST_NAME	JOB_ID
100	Steven	AD_VP

Cuarto punto:

En el último punto se nos pedía realizar un disparador (trigger) que detecte cuándo se ha modificado el salario de una persona y en caso de tener un salario distinto en la modificación, incluya un registro dentro de una tabla creado a tal efecto. A continuación presentamos cómo el Trigger muestra su correcto funcionamiento en el Live SQL.

```sql
CREATE TABLE emp_audit(
    id_employee INTEGER,
    momento_actualizacion TIMESTAMP,
    log VARCHAR2(200)
);

--Ahora que ya está creada, ya podemos crear nuestro trigger.

CREATE OR REPLACE TRIGGER disparador_cambio_salario
    -- Como le decimos que es la tabla employees, ya no hay que especificarlo después.
    AFTER UPDATE OF salary ON employees
    FOR EACH ROW
DECLARE --el bloque declare no es necesario como tal, es por comodidad
    mensaje VARCHAR2(200);
BEGIN
    IF :old.salary != :new.salary THEN
        mensaje := '-Salario anterior: ' || :old.salary || ' -Salario nuevo: ' || :new.salary;

        --  Y ahora procedemos con los cambios
        INSERT INTO emp_audit
        VALUES (
        :old.employee_id,
        SYSTIMESTAMP,
        mensaje
        );
    ELSE
        NULL;
    END IF;
END;
```

```
172 /*Finalmente, podemos ejecutar la sentencia correspondiente para ver qué sucede: */
173 UPDATE employees SET salary=18000 WHERE employee_id=103;
174 SELECT * FROM emp_audit;
175
176 --Si volviésemos a ejecutar la misma sentencia, no sucedería nada en absoluto.
177
```

ID_EMPLOYEE	MOMENTO_ACTUALIZACION	LOG
103	09-MAY-24 04.56.43.455108 PM	-Salario anterior: 9000 -Salario nuevo: 18000

5. Puesta en común

Cada uno de nosotros ha realizado su proyecto individualmente usando la plataforma GitHub algunos de nosotros como comentamos al principio del documento y otros pasando su proyecto por zip vía WhatsApp. Posteriormente hemos realizado cuatro reuniones en las que hemos debatido y probado los diferentes códigos para poder solucionar los errores que nos fuesen saliendo y documentar el código.

También hemos usado WhatsApp para organizar las reuniones y para poder irnos ayudando cuando alguno de nosotros lo necesitaba para poder seguir realizando su proyecto.