**UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS**
**FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA**
**ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS**


# “LABORATORIO 06”

**ASIGNATURA**

Administración de Base de Datos

**DOCENTE**

Chávez Soto, Jorge Luis

**ESTUDIANTE**

Carhuaricra Anco, Heidy Nicole - 23200150

**Lima, Perú**

**2025**

# ÍNDICE

# LABORATORIO 06

## I. OBJETIVOS

El presente laboratorio tiene por objetivos:

- Aplicar los comandos COMMIT, ROLLBACK y SAVEPOINT para controlar la persistencia de los datos.
- Simular escenarios de bloqueos y concurrencia entre sesiones de Oracle.
- Garantizar la consistencia y atomicidad de las operaciones de actualización.
- Comprender el impacto de las transacciones en la base de datos HR durante procesos de inserción, actualización y eliminación.

## II. DESCRIPCIÓN DE LAS TABLAS

- ***Countries***

| Name | Null? | Type |
|---|---|---|
| COUNTRY_ID | NOT NULL | CHAR (2) |
| COUNTRY_NAME | | VARCHAR2 (40) |
| REGION_ID | | NUMBER |

- ***Departments***

| Name | Null? | Type |
|---|---|---|
| DEPARTMENT_ID | NOT NULL | NUMBER (4) |
| DEPARTMENT_NAME | NOT NULL | VARCHAR2 (30) |
| MANAGER_ID | | NUMBER (5) |
| LOCATION_ID | | NUMBER (4) |

- ***Employees***

| Name | Null? | Type |
|---|---|---|
| EMPLOYEE_ID | NOT NULL | NUMBER (6) |
| FIRST_NAME | | VARCHAR2 (20) |
| LAST_NAME | NOT NULL | VARCHAR2 (25) |
| EMAIL | NOT NULL | VARCHAR2 (25) |
| PHONE_NUMBER | | VARCHAR2 (20) |
| HIRE_DATE | NOT NULL | DATE |
| JOB_ID | NOT NULL | VARCHAR2 (10) |

| | | |
|---|---|---|
| SALARY | | NUMBER (8, 2) |
| COMMISSION_PCT | | NUMBER (2, 2) |
| MANAGER_ID | | NUMBER (6) |
| DEPARTMENT_ID | | NUMBER (4) |

- ***Jobs***

| Name | Null? | Type |
|---|---|---|
| JOB_ID | NOT NULL | VARCHAR2 (10) |
| JOB_TITLE | NOT NULL | VARCHAR2 (35) |
| MIN_SALARY | | NUMBER (6) |
| MAX_SALARY | | NUMBER (6) |

- ***Job_History***

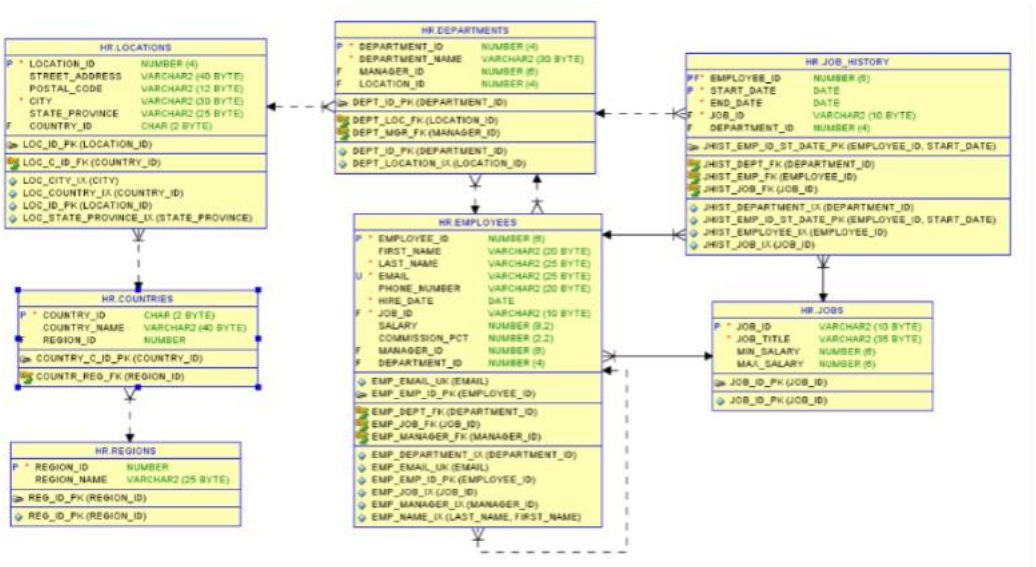
| Name | Null? | Type |
|---|---|---|
| EMPLOYEE_ID | NOT NULL | NUMBER (6) |
| START_DATE | NOT NULL | DATE |
| END_DATE | NOT NULL | DATE |
| JOB_ID | NOT NULL | VARCHAR2 (10) |
| DEPARTMENT_ID | | NUMBER (4) |

- ***Locations***

| Name | Null? | Type |
|---|---|---|
| LOCATION_ID | NOT NULL | NUMBER (4) |
| STREET_ADDRESS | | VARCHAR2 (40) |
| POSTAL_CODE | | VARCHAR2 (12) |
| CITY | NOT NULL | VARCHAR2 (30) |
| STATE PROVINCE | | VARCHAR2 (25) |
| COUNTRY_ID | | CHAR (2) |

- ***Regions***

| Name | Null? | Type |
|---|---|---|
| RGION_ID | NOT NULL | NUMBER () |
| REGION_NAME | | VARCHAR2 (25) |




## III. EJERCICIOS PLANTEADOS

### *1. Control básico de transacciones*

Cree un bloque anónimo PL/SQL que:

- Aumente en un 10% el salario de los empleados del departamento 90.
- Guarde un `SAVEPOINT` llamado punto1.
- Aumente en un 5% el salario de los empleados del departamento 60.
- Realice un `ROLLBACK` al `SAVEPOINT` punto1.
- Ejecute finalmente un `COMMIT`.

Preguntas:

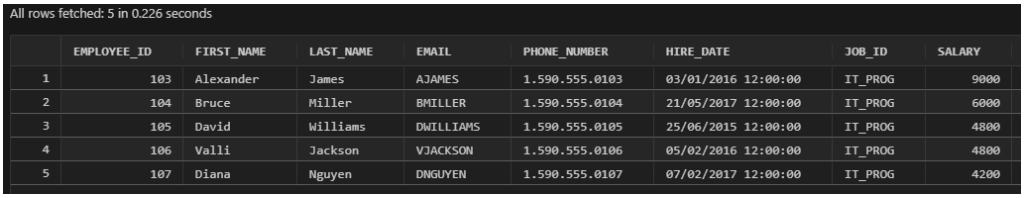
a. ¿Qué departamento mantuvo los cambios?
b. ¿Qué efecto tuvo el `ROLLBACK` parcial?
c. ¿Qué ocurriría si se ejecutara `ROLLBACK` sin especificar `SAVEPOINT`?

***Solución:***

```
--- Ejercicio 01
--- Consultas necesarias
SELECT * FROM employees WHERE department_id = 90;
SELECT * FROM employees WHERE department_id = 60;
```

All rows fetched: 3 in 0.225 seconds

| | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE | JOB_ID | SALARY | COMMISSION_PCT |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 100 | Steven | King | SKING | 1.515.555.0100 | 17/06/2013 12:00:00 | AD_PRES | 24000 | (null) |
| 2 | 101 | Neena | Yang | NYANG | 1.515.555.0101 | 21/09/2015 12:00:00 | AD_VP | 17000 | (null) |
| 3 | 102 | Lex | Garcia | LGARCIA | 1.515.555.0102 | 13/01/2011 12:00:00 | AD_VP | 17000 | (null) |



All rows fetched: 5 in 0.226 seconds

| | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE | JOB_ID | SALARY |
|---|---|---|---|---|---|---|---|---|
| 1 | 103 | Alexander | James | AJAMES | 1.590.555.0103 | 03/01/2016 12:00:00 | IT_PROG | 9000 |
| 2 | 104 | Bruce | Miller | BMILLER | 1.590.555.0104 | 21/05/2017 12:00:00 | IT_PROG | 6000 |
| 3 | 105 | David | Williams | DWILLIAMS | 1.590.555.0105 | 25/06/2015 12:00:00 | IT_PROG | 4800 |
| 4 | 106 | Valli | Jackson | VJACKSON | 1.590.555.0106 | 05/02/2016 12:00:00 | IT_PROG | 4800 |
| 5 | 107 | Diana | Nguyen | DNGUYEN | 1.590.555.0107 | 07/02/2017 12:00:00 | IT_PROG | 4200 |

```
--- Bloque anónimo PL/SQL
DECLARE
BEGIN
    --- Aumento del salario al 10% a empleados del dpto 90
    UPDATE employees
    SET salary = salary * 1.1
    WHERE department_id = 90;

    --- Crear un punto de guardado
    SAVEPOINT aumento1;

    --- Aumento del salario al 5% a empleados del dpto 60
    UPDATE employees
    SET salary = salary * 1.05
    WHERE department_id = 60;

    --- Deshacer cambios realizados desde el savepoint en
adelante
    ROLLBACK TO aumento1;

    COMMIT;
END;
/
```

Consultamos nuevamente para revisar las actualizaciones:

```
--- Consultas necesarias
SELECT * FROM employees WHERE department_id = 90;
SELECT * FROM employees WHERE department_id = 60;
```

All rows fetched: 3 in 0.155 seconds

| | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE | JOB_ID | SALARY |
|---|---|---|---|---|---|---|---|---|
| 1 | 100 | Steven | King | SKING | 1.515.555.0100 | 17/06/2013 12:00:00 | AD_PRES | 26400 |
| 2 | 101 | Neena | Yang | NYANG | 1.515.555.0101 | 21/09/2015 12:00:00 | AD_VP | 18700 |
| 3 | 102 | Lex | Garcia | LGARCIA | 1.515.555.0102 | 13/01/2011 12:00:00 | AD_VP | 18700 |

All rows fetched: 5 in 0.226 seconds

| | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE | JOB_ID | SALARY |
|---|---|---|---|---|---|---|---|---|
| 1 | 103 | Alexander | James | AJAMES | 1.590.555.0103 | 03/01/2016 12:00:00 | IT_PROG | 9000 |
| 2 | 104 | Bruce | Miller | BMILLER | 1.590.555.0104 | 21/05/2017 12:00:00 | IT_PROG | 6000 |
| 3 | 105 | David | Williams | DWILLIAMS | 1.590.555.0105 | 25/06/2015 12:00:00 | IT_PROG | 4800 |
| 4 | 106 | Valli | Jackson | VJACKSON | 1.590.555.0106 | 05/02/2016 12:00:00 | IT_PROG | 4800 |
| 5 | 107 | Diana | Nguyen | DNGUYEN | 1.590.555.0107 | 07/02/2017 12:00:00 | IT_PROG | 4200 |

**¿Qué departamento mantuvo los cambios?**

El departamento 90 mantuvo el aumento del 10% en el salario, porque el `ROLLBACK` solo afectó los cambios posteriores al `SAVEPOINT`.

**¿Qué efecto tuvo el `ROLLBACK` parcial?**

El `ROLLBACK` deshizo únicamente las operaciones ejecutadas después del `SAVEPOINT aumento1`, en este caso el aumento del 5% del departamento 60. Los cambios anteriores (departamento 90) permanecieron activos.

**¿Qué ocurriría si se ejecutara `ROLLBACK` sin especificar `SAVEPOINT`?**

Se desharían todos los cambios de la transacción completa, incluyendo el aumento del 10% del salario al departamento 90. En ese caso, nungún cambio sería guardado, incluso si luego se hiciera `COMMIT`.

## 2. *Bloqueos entre sesiones*

En dos sesiones diferentes de Oracle:

- En la primera sesión, ejecute: `UPDATE employees SET salary = salary + 500 WHERE employee_id = 103;`
- Sin ejecutar `COMMIT`, en la segunda sesión, intente modificar el mismo registro.
- Observe el bloqueo y, desde la primera sesión, ejecute: `ROLLBACK;`
- Analice el efecto sobre la segunda sesión.

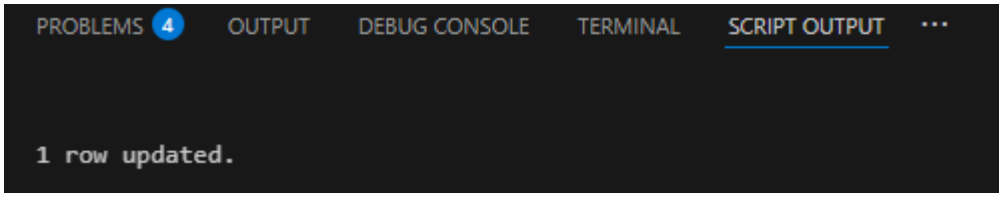
Preguntas:

a. ¿Por qué la segunda sesión quedó bloqueada?
b. ¿Qué comando libera los bloqueos?
c. ¿Qué vistas del diccionario permiten verificar sesiones bloqueadas?

***Solución:***

```
--- Ejercicio 02: Sesión 1
--- Consulta necesaria
SELECT employee_id, salary FROM employees WHERE employee_id =
103;
```


```
--- Actualización de salario
UPDATE employees
SET salary = salary + 500
WHERE employee_id = 103;
```

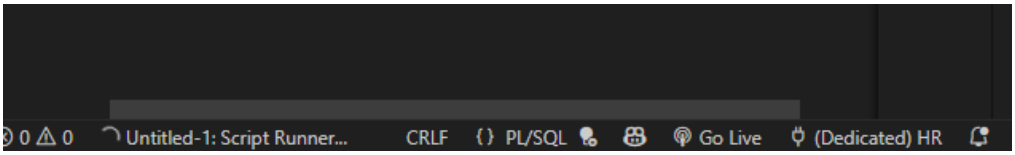


Ejecutamos la actualización de salario en la sesión 2

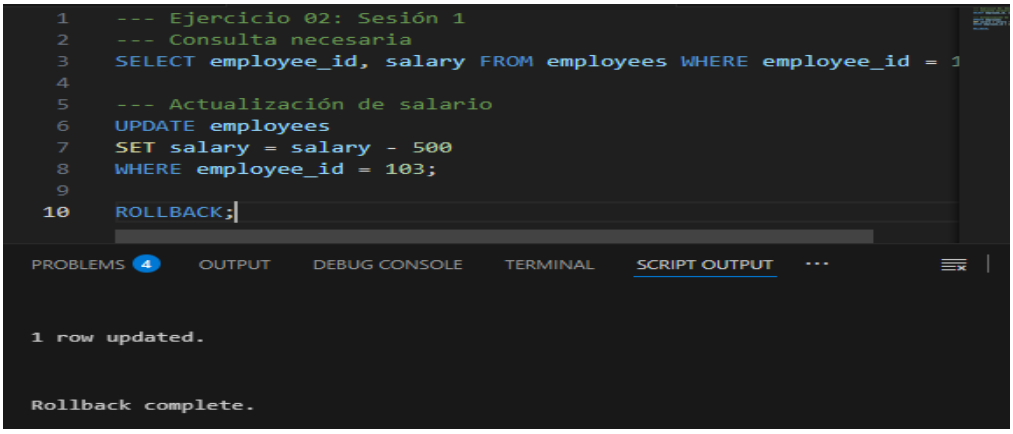
```
--- Ejercicio 02: Sesión 2
--- Consulta necesaria
SELECT employee_id, salary FROM employees WHERE employee_id =
103;

--- Actualización de salario
UPDATE employees
SET salary = salary + 200
WHERE employee_id = 103;
```

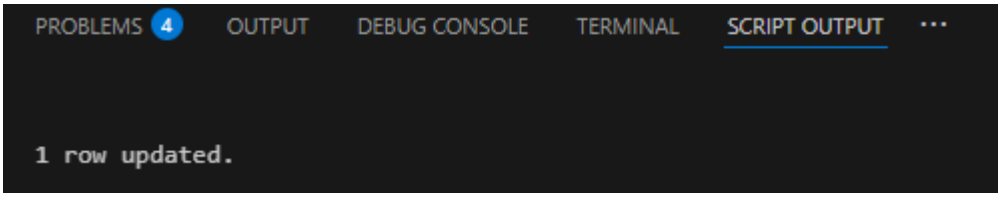
No termina de ejecutar el UPDATE




Ejecutamos un ROLLBACK en la sesión 1




Automáticamente, se termina de ejecutar el UPDATE en la sesión 2




**¿Por qué la segunda sesión quedó bloqueada?**

La segunda sesión quedó bloqueada porque Oracle usa bloqueos implícitos a nivel de fila (row – level locks) cuando una transacción modifica datos.

La primera sesión tiene un bloqueo exclusivo (Exclusive Lock) sobre la fila del `employee_id = 103` para evitar inconsistencias. Mientras el `UPDATE` no ejecute un `COMMIT` o un `ROLLBACK`, ninguna otra sesión puede modificar esa misma fila. Por ello, la segunda sesión queda en espera (bloqueada) hasta que se libere el recurso.

**¿Qué comando libera los bloqueos?**

Los bloqueos se liberan automáticamente al ejecutar no de los siguientes comandos:

- `COMMIT` -> Confirma los cambios y libera los locks.
- `ROLLBACK` -> Revierte los cambios y también libera los locks.

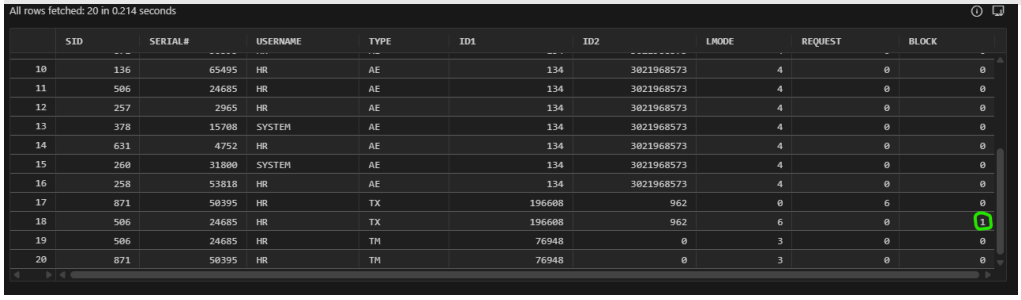
**¿Qué vistas del diccionario permiten verificar sesiones bloqueadas?**

Se puede consultar las siguientes vistas en el diccionario de Oracle para analizar bloqueos:

| Vista | Descripción |
|---|---|
| `V$LOCK` | Muestra los bloqueos actuales en el sistema (tipo de lock, modo, identificadores de sesión). |
| `V$SESSION` | Muestra las sesiones activas; se puede combinar con V$LOCK para identificar quién bloquea a quién. |
| `DBA_BLOCKERS` | Muestra las sesiones que están bloqueando a otras. |
| `DBA_WAITERS` | Muestra las sesiones que están esperando a ser desbloqueadas. |
| `V$LOCKED_OBJECT` | Relaciona los objetos (tablas, filas) que están bloqueados con las sesiones que los poseen. |

Realizamos la siguiente consulta en el diccionario de datos para saber quiénes interactúan en el bloqueo:

```
SELECT
    s.sid,
    s.serial#,
    s.username,
    l.type,
    l.id1,
    l.id2,
    l.lmode,
    l.request,
    l.block
FROM v$session s
JOIN v$lock l ON s.sid = l.sid
WHERE s.username IS NOT NULL;
```


All rows fetched: 20 in 0.214 seconds

| | SID | SERIAL# | USERNAME | TYPE | ID1 | ID2 | LMODE | REQUEST | BLOCK |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 136 | 65495 | HR | AE | 134 | 3021968573 | 4 | 0 | 0 |
| 11 | 506 | 24685 | HR | AE | 134 | 3021968573 | 4 | 0 | 0 |
| 12 | 257 | 2965 | HR | AE | 134 | 3021968573 | 4 | 0 | 0 |
| 13 | 378 | 15708 | SYSTEM | AE | 134 | 3021968573 | 4 | 0 | 0 |
| 14 | 631 | 4752 | HR | AE | 134 | 3021968573 | 4 | 0 | 0 |
| 15 | 260 | 31800 | SYSTEM | AE | 134 | 3021968573 | 4 | 0 | 0 |
| 16 | 258 | 53818 | HR | AE | 134 | 3021968573 | 4 | 0 | 0 |
| 17 | 871 | 50395 | HR | TX | 196608 | 962 | 0 | 6 | 0 |
| 18 | 506 | 24685 | HR | TX | 196608 | 962 | 6 | 0 | 1 |
| 19 | 506 | 24685 | HR | TM | 76948 | 0 | 3 | 0 | 0 |
| 20 | 871 | 50395 | HR | TM | 76948 | 0 | 3 | 0 | 0 |

### *3. Transacción controlada con bloque PL/SQL*

Cree un bloque anónimo PL/SQL que realice una transferencia de empleado de un departamento a otro, registrando la transacción en `JOB_HISTORY`.

Pasos:

- Actualice el `department_id` del empleado 104 al departamento 110.
- Inserte simultáneamente el registro correspondiente en `JOB_HISTORY`.
- Si ocurre un error (por ejemplo, departamento inexistente), haga un `ROLLBACK` y muestre un mensaje con `DBMS_OUTPUT`.

Preguntas:

a. ¿Por qué se debe garantizar la atomicidad entre las dos operaciones?
b. ¿Qué pasaría si se produce un error antes del `COMMIT`?
c. ¿Cómo se asegura la integridad entre `EMPLOYEES` y `JOB_HISTORY`?

***Solución:***

```
--- Ejercicio 03
SET serveroutput ON;
DECLARE
    v_employee_id       employees.employee_id%TYPE := 104;
    v_antiguo_dpto      employees.department_id%TYPE;
    v_nuevo_dpto        employees.department_id%TYPE := 110;
BEGIN
    --- Obtener el departamento actual del empleado
    SELECT department_id
        INTO v_antiguo_dpto
    FROM employees
    WHERE employee_id = v_employee_id;

    --- Actualizar el departamento del empleado
    UPDATE employees
    SET department_id = v_nuevo_dpto
    WHERE employee_id = v_employee_id;

    --- Insertar registro en JOB_HISTORY
    INSERT INTO job_history (employee_id, start_date, end_date,
job_id, department_id)
    VALUES (v_employee_id, SYSDATE - 30, SYSDATE, 'SA_REP',
v_antiguo_dpto);

    --- Confirmar transacción
    COMMIT;
      dbms_output.put_line('Transferencia             realizada
correctamente.');
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        ROLLBACK;
        dbms_output.put_line('Error: No existe el empleado o el
departamento especificado.');
    WHEN OTHERS THEN
```

```
        ROLLBACK;
        dbms_output.put_line('Error     al     realizar     la
transferencia: ' || SQLERRM);
END;
/
```

```
Transferencia realizada correctamente.


PL/SQL procedure successfully completed.
```

**¿Por qué se debe garantizar la atomicidad entre las dos operaciones?**

Porque las acciones realizadas (actualizar departamento y registrar el cambio en `JOB_HISTORY`) forman parte de una misma transacción lógica. Si una se realiza sin la otra, los datos quedarían inconsistentes. Por ello, la atomicidad se basa en ejecutar ambas acciones o ninguna.

**¿Qué pasaría si se produce un error antes del `COMMIT`?**

Si ocurre un error antes del `COMMIT`, y el bloque ejecuta un `ROLLBACK` en la sección `EXCEPTION`, es decir, todas las operaciones previas se revierten automáticamente.

**¿Cómo se asegura la integridad entre `EMPLOYEES` y `JOB_HISTORY`?**

Oracle mantiene esta integridad mediante:

- *Claves foráneas (FOREIGN KEY)* `JOB_HISTORY.EMPLOYEE_ID` hace referencia a `EMPLOYEES.EMPLOYEE_ID`. Esto impide insertar un historial de un empleado que no exista.
- *Lógica transaccional (COMMIT/ROLLBACK)* El bloque PL/SQL asegura que ambas tablas se actualicen en conjunto.
- Restricciones de integridad referencial La base de datos garantiza que los valores en `JOB_HISTORY.DEPARTMENT_ID` existan en `DEPARTMENTS.DEPARTMENT_ID`.

## *4. SAVEPOINT y reversión parcial*

Diseñe un bloque anónimo PL/SQL que ejecute las siguientes operaciones en una sola transacción:

- Aumentar el salario en 8% para empleados del departamento 100 → `SAVEPOINT A`.
- Aumentar el salario en 5% para empleados del departamento 80 → `SAVEPOINT B`.
- Eliminar los empleados del departamento 50.
- Revierte los cambios hasta el `SAVEPOINT B`.
- Finalmente, confirma la transacción con `COMMIT`.

Preguntas:

a. ¿Qué cambios quedan persistentes?
b. ¿Qué sucede con las filas eliminadas?
c. ¿Cómo puedes verificar los cambios antes y después del `COMMIT`?

***Solución:***

Realizamos la(s) consulta(s) correspondiente(s):

```
--- Ejercicio 04
SELECT employee_id, salary, department_id
FROM employees
WHERE department_id IN (50, 80, 100);
```


| | EMPLOYEE_ID | SALARY | DEPARTMENT_ID |
|---|---|---|---|
| 1 | 199 | 2600 | 50 |
| 2 | 108 | 12008 | 100 |
| 3 | 109 | 9000 | 100 |
| 4 | 110 | 8200 | 100 |
| 5 | 111 | 7700 | 100 |
| 6 | 112 | 7800 | 100 |
| 7 | 113 | 6900 | 100 |

```
--- Bloque anónimo PL/SQL
DECLARE
BEGIN
    --- Aumento del 8% para empleados del departamento 100
    UPDATE employees
        SET salary = salary * 1.08
    WHERE department_id = 100;

    SAVEPOINT aumento1;
    dbms_output.put_line('Aumento del 8% aplicado al
departamento 100.');

    --- Aumento del 5% para empleados del departamento 80
```

```
    UPDATE employees
        SET salary = salary * 1.05
    WHERE department_id = 80;

    SAVEPOINT aumento2;
    dbms_output.put_line('Aumento    del    5%    aplicado    al
departamento 80.');

    --- Eliminación de empleados del departamento 50
    DELETE FROM employees
    WHERE department_id = 50;
    dbms_output.put_line('Empleados    del    departamento    50
eliminados.');

    --- Reversión parcial: volver al SAVEPOINT aumento2
    ROLLBACK TO SAVEPOINT aumento2;
    dbms_output.put_line('Cambios revertidos hasta el SAVEPOINT
aumento2.');

    --- Confirmar la transacción
    COMMIT;
    dbms_output.put_line('Transacción confirmada.');
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        dbms_output.put_line('Error: ' || SQLERRM);
END;
/
```

```
Aumento del 8% aplicado al departamento 100.
Aumento del 5% aplicado al departamento 80.
Error: ORA-02292: integrity constraint (HR.DEPT_MGR_FK) violated - child record f


PL/SQL procedure successfully completed.
```

**¿Qué cambios quedan persistentes?**

Ningún cambio queda persistente. El error `ORA-02292` provocó que la transacción no pudiera completarse, y como el bloque ejecutó un `ROLLBACK` parcial o total, todos los cambios anteriores (aumentos del 8% y 5%) fueron revertidos antes del `COMMIT`.

**¿Qué sucede con las filas eliminadas?**

Las filas no se llegaron a eliminar. El intento de `DELETE FROM employees WHERE department_id = 50` falló porque existian registros dependientes (clave foránea violada). Oracle nunca elimina parcialmente si hay error de integridad referencial: simplemente anula toda la operación.

### ¿Cómo puedes verificar los cambios antes y después del `COMMIT`?

Para verificar los efectos de una transacción, se pueden realizar consultas antes y después de ejecutar las sentencias `COMMIT` o `ROLLBACK`.

Antes del `COMMIT` o `ROLLBACK`, es posible visualizar los cambios provisionales que aún no han sido confirmados. Después de ejecutar las sentencias, los resultados mostrados reflejarán el estado real y confirmado de la base de datos. De esta forma, se puede comprobar qué operaciones fueron finalmente persistidas o revertidas según el control de transacciones ejecutado.