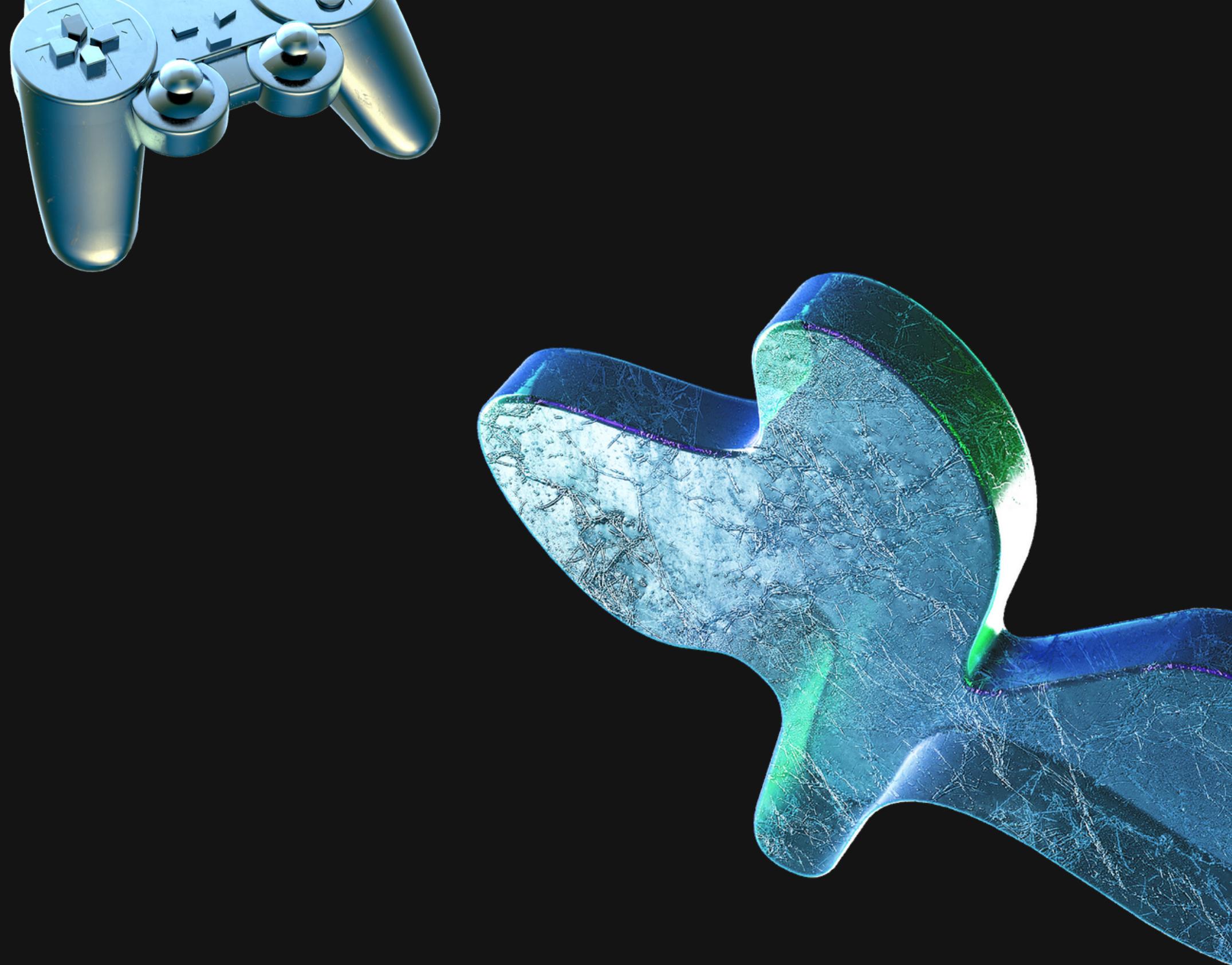


# PL-SQL



# Tópicos



Procedimientos y Funciones Locales

---

Procedimientos y Funciones  
Almacenadas

---

Locales vs almacenados

---

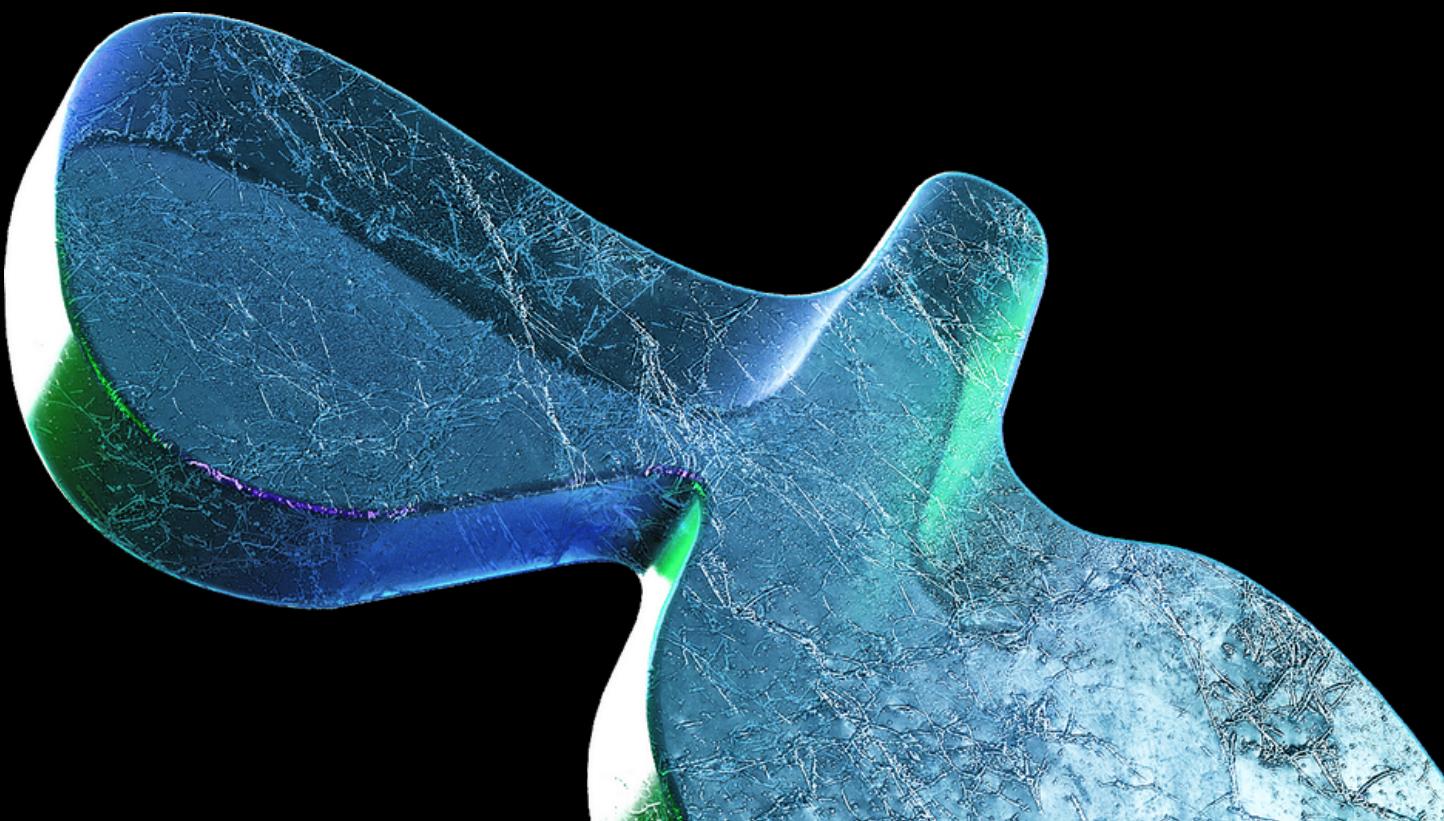
Paquetes

---

Triggers

---

# Procedimientos y Funciones Locales



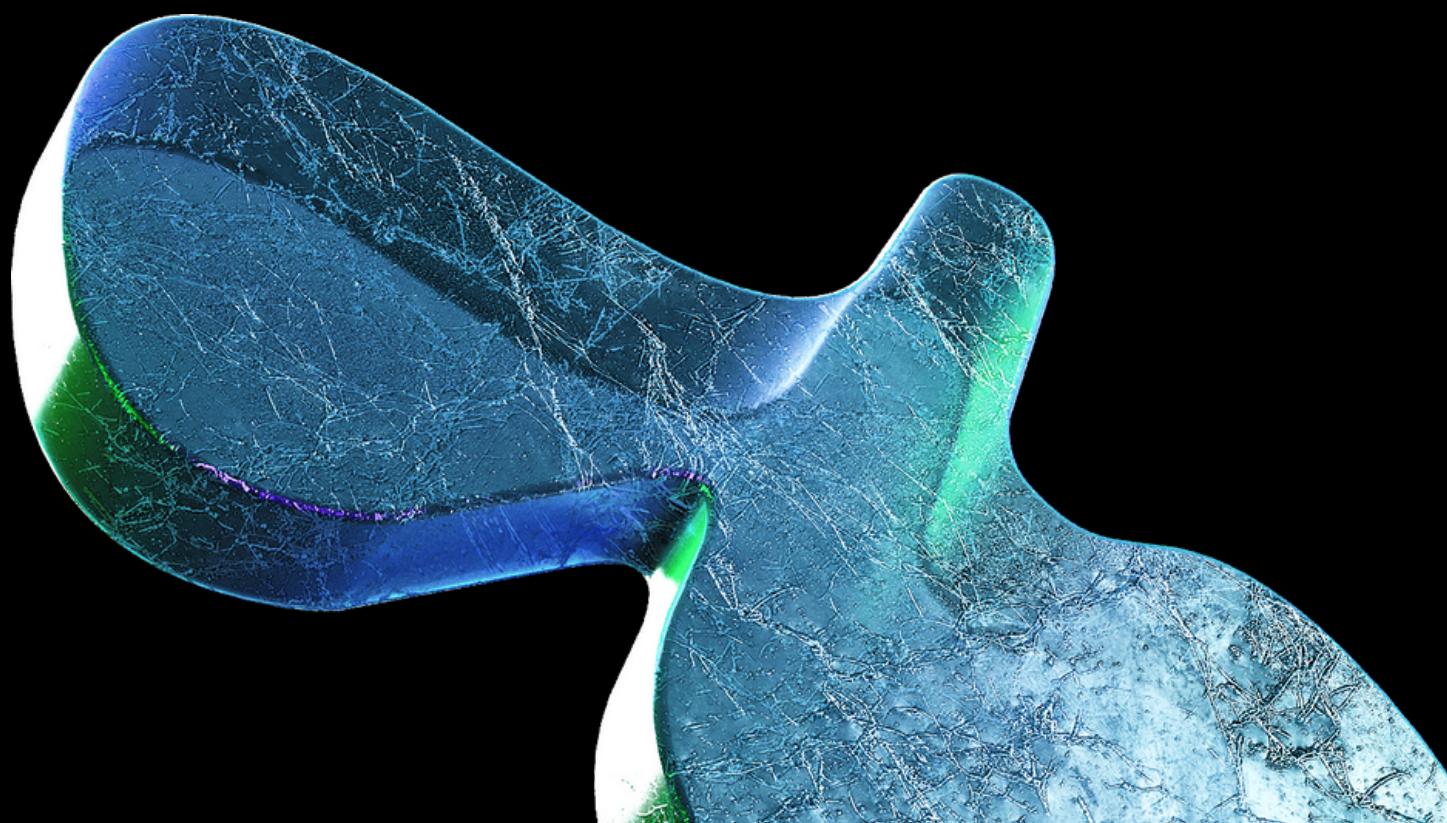
**DECLARE**

```
...
PROCEDURE ... IS
BEGIN
[EXCEPTION]
END;
```

**BEGIN**

```
...
Procedure_name ... ;
...
...
```

# Procedimientos y Funciones Locales



## Sintaxis Cuerpo del procedimiento:

```
PROCEDURE nombre_procedimiento [ (parámetro [, parámetro, . . . ]) ] IS
    [declaraciones locales]
BEGIN
    Sentencias ejecutables
[EXCEPTION
    manejadores de excepciones]
END [nombre_procedimiento];
```

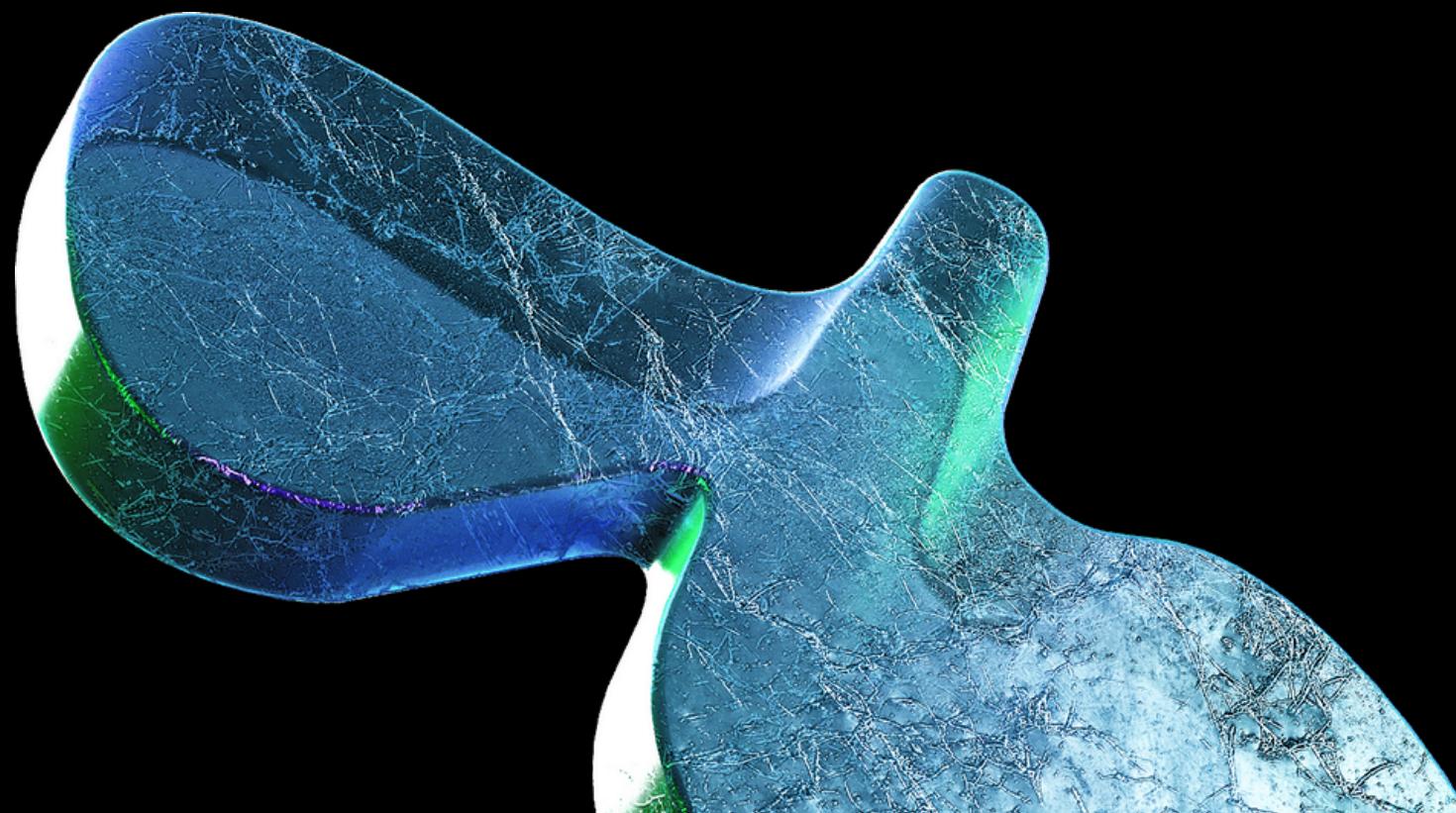
donde: *parámetro* representa lo siguiente:

*nombre\_parametro* modo tipo de dato [: = *value*]

## Sintaxis Llamado a un procedimiento:

```
nombre_procedimiento [ ( nombre_parametro [, nombre_parametro, . . . ] ) ]
```

# Procedimientos y Funciones Locales



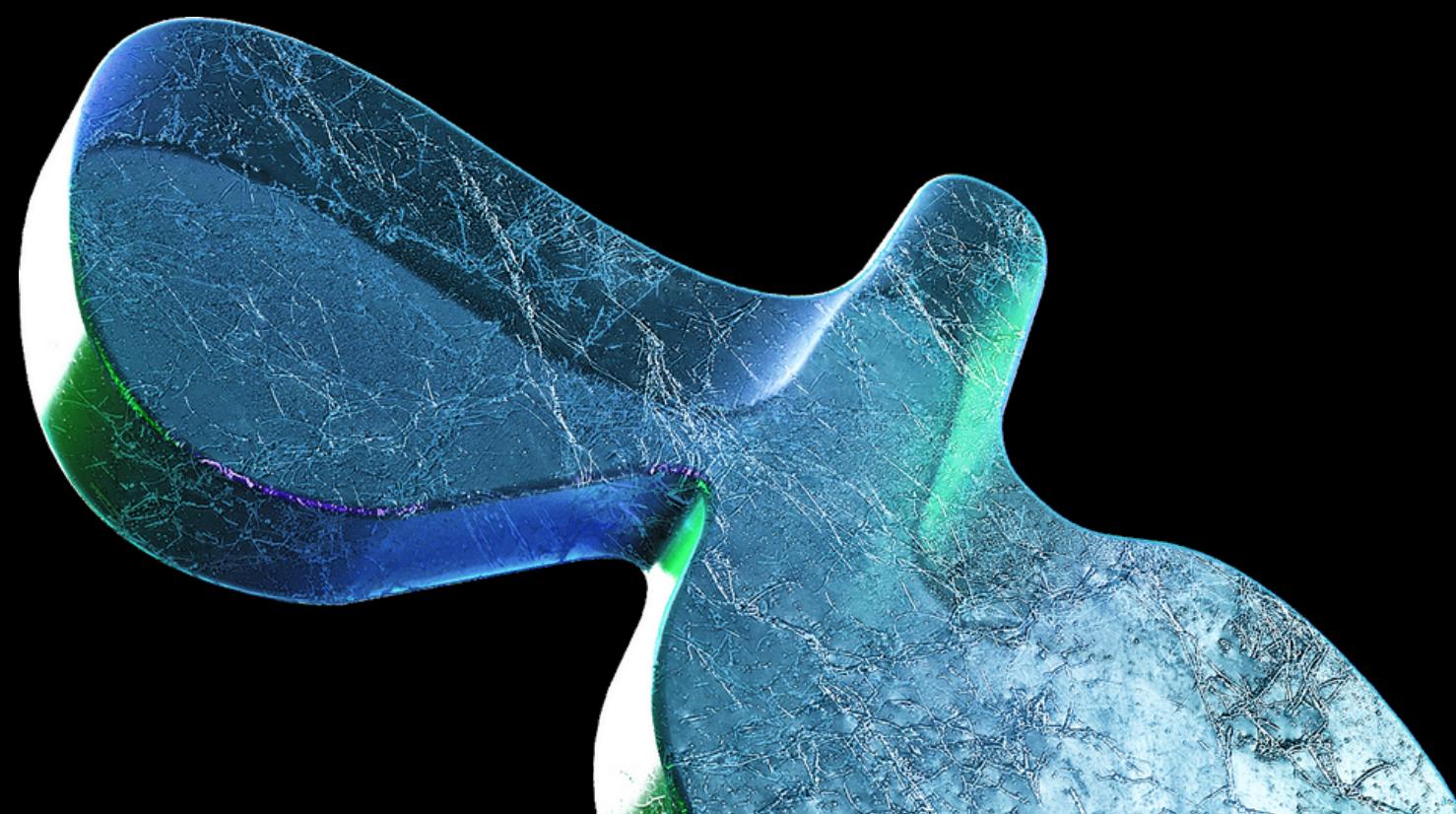
**Ejemplo:** Llamada a un procedimiento con parámetros IN

raise\_salary (emp\_num, amount);

**Cuerpo del procedimiento** con parámetros IN

```
PROCEDURE raise_salary (emp_id IN INTEGER, increase IN REAL) IS
    current_salary REAL;
    salary_missing EXCEPTION;
BEGIN
    SELECT sal INTO current_salary FROM emp
        WHERE empno = emp_id;
    IF current_salary IS NULL THEN
        RAISE salary_missing;
    ELSE
        UPDATE emp SET sal = sal + increase
            WHERE empno = emp_id;
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        INSERT INTO emp_audit VALUES (emp_id, 'No such number');
    WHEN salary_missing THEN
        INSERT INTO emp_audit VALUES (emp_id, 'salary is null');
END raise_salary;
```

# Procedimientos y Funciones Locales



**Paso de información entre procedimientos usando parámetros**

**Sintaxis del uso de parámetros:**

**PROCEDURE *nombre\_procedimiento* (*nombre\_parametros***

**modo tipo de dato [:= value])**

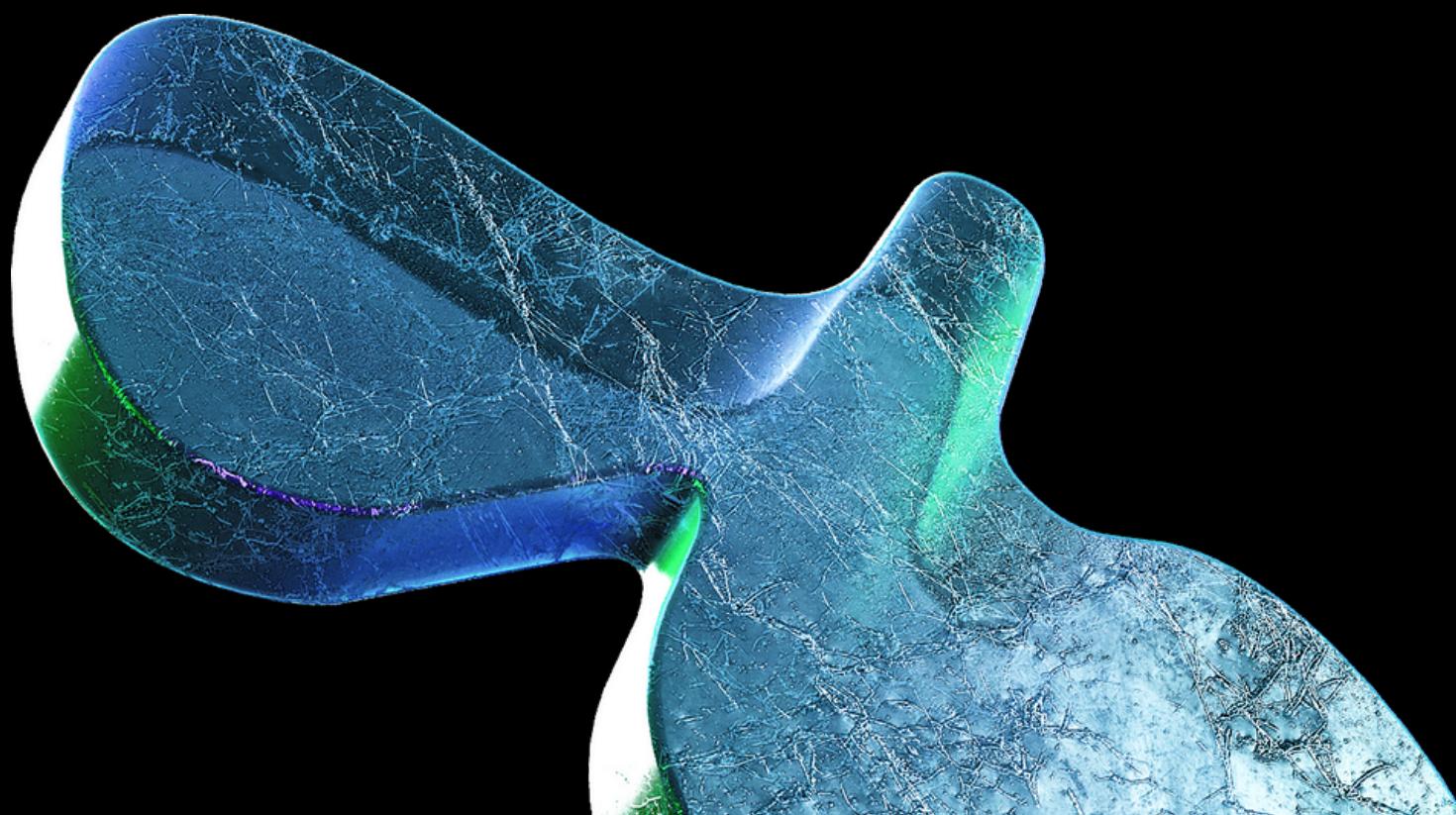
donde: **modo** puede ser IN , OUT, IN OUT.

**IN:** Permite el traspaso de valores a un procedimiento. Actúa como una constante y no se le puede asignar un valor en el procedimiento. IN es el modo por defecto.

**OUT:** Permite retornar valores hacia el que llama al procedimiento. Actúa como una variable sin inicializar y no puede ser asignada a otra variable o reasignada a ella misma en el procedimiento.

**IN OUT:** Permite el traspaso de valores iniciales a un procedimiento y retorna valores actualizados al llamador. Actúa como una variable inicializada.

# Procedimientos y Funciones Locales



**FUNCTION ... IS**

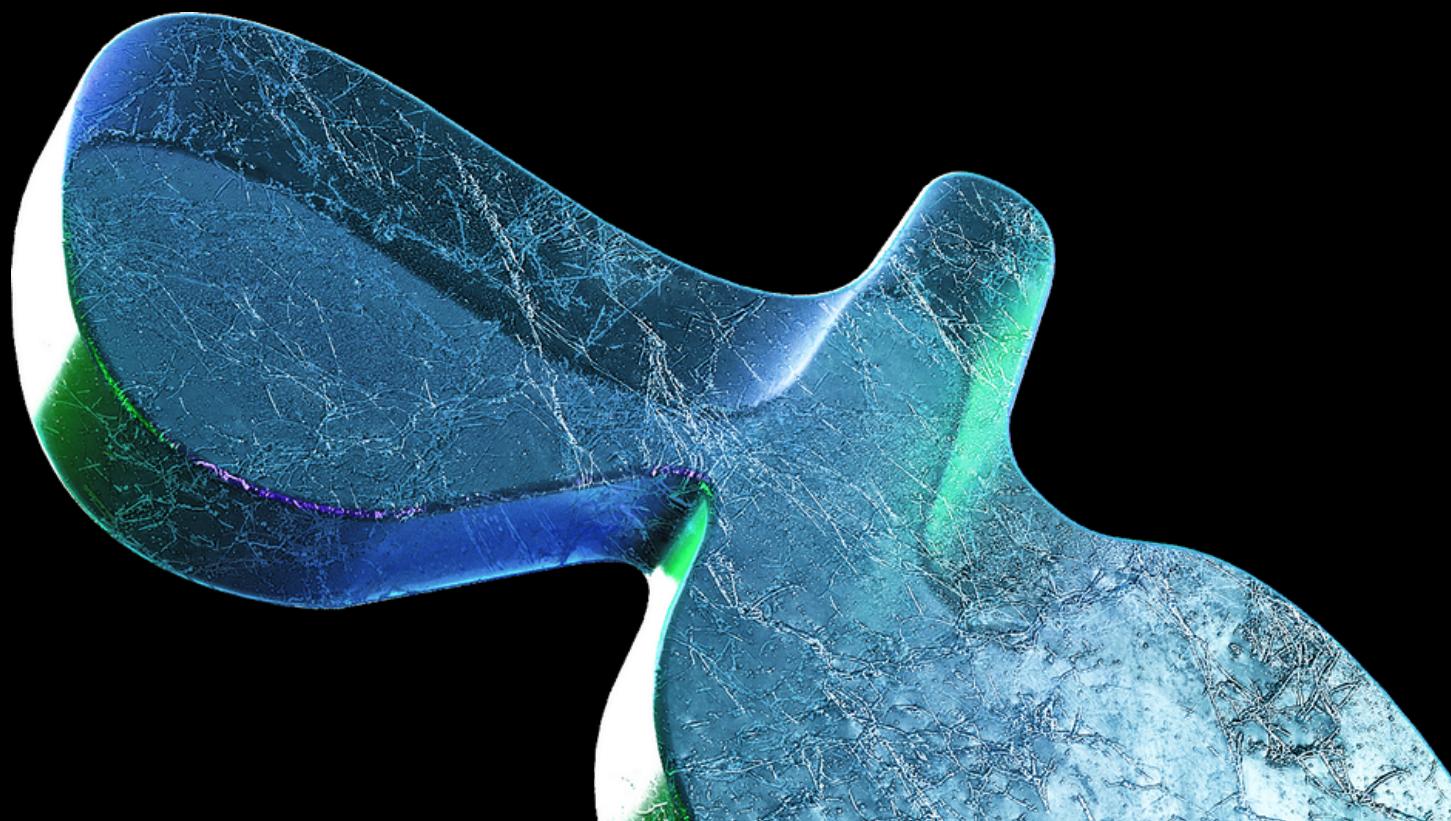
**BEGIN**

*Tiene que tener un return*

**[EXCEPTION]**

**END;**

# Procedimientos y Funciones Locales



## Sintaxis Cuerpo de la función:

```
FUNCTION nombre_funcion [(parametro [, parametro, ...]) ]  
    RETURN datatype IS  
        [declaraciones locales]  
BEGIN  
        Sentencias ejecutables  
[EXCEPTION  
        manejadores de excepciones]  
END [nombre_funcion];
```

donde: *parametro* representa lo siguiente:

*nombre\_parametro* modo tipo de dato [: = value]

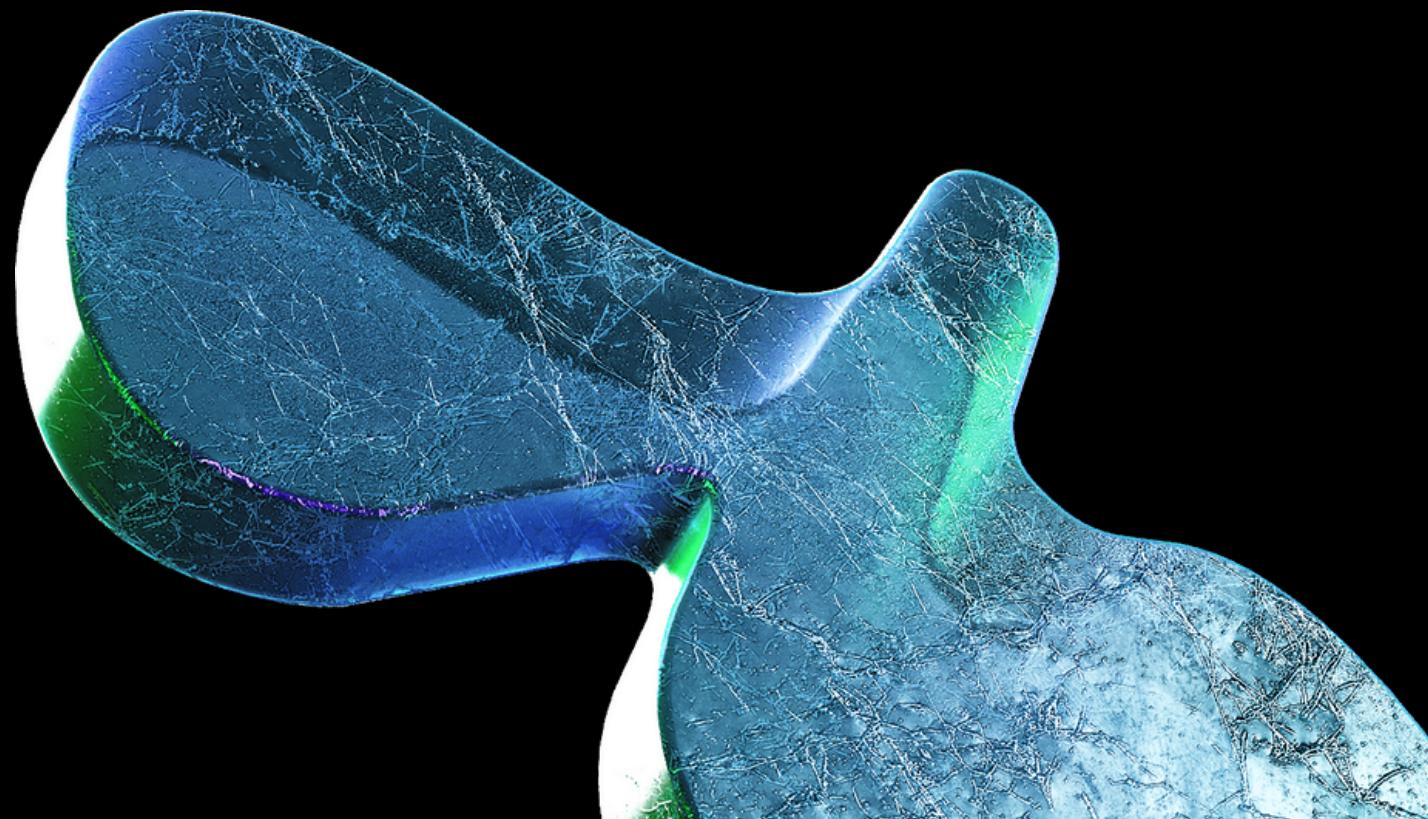
## Sintaxis Llamado a una función como parte de una expresión:

```
... nombre_funcion [ (nombre_parametro [, nombre_parametro, ...] ) ] ...
```

## Debe considerarse:

- Una función debe ser llamada como parte de una expresión.
- Los llamados a funciones definidas por el usuario, pueden aparecer en sentencias procedurales, pero no en sentencias SQL.

# Procedimientos y Funciones Locales



## Ejemplo: Cuerpo de función

```
FUNCTION sal_ok (salary IN REAL, level IN CHAR)
  RETURN BOOLEAN IS
    min_sal  REAL;
    max_sal  REAL;
BEGIN
  SELECT losal, hisal INTO min_sal, max_sal
  FROM salgrade
  WHERE grade = level;
  RETURN (salary >= min_sal) AND (salary <= max_sal);
END sal_ok;
```

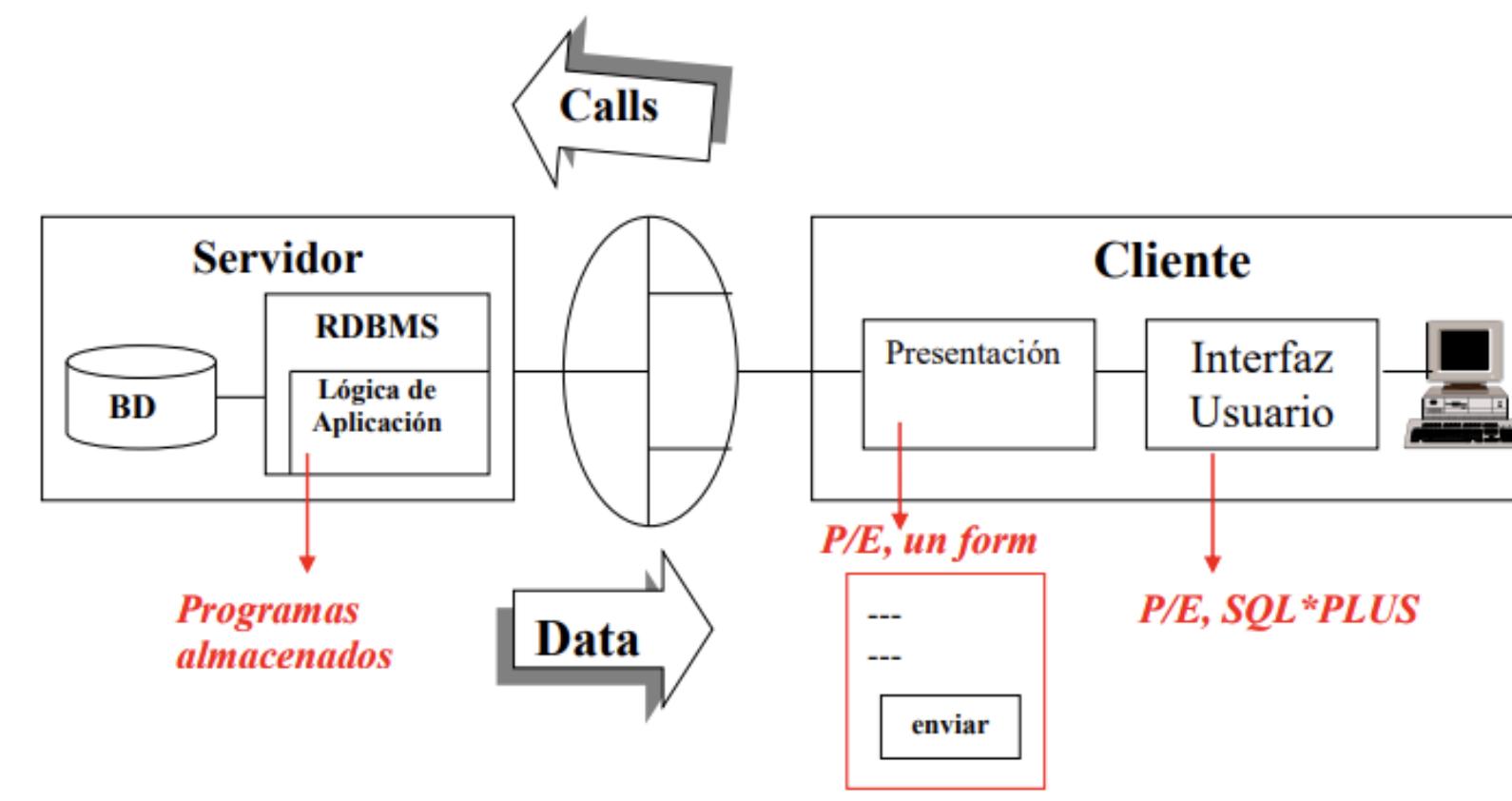
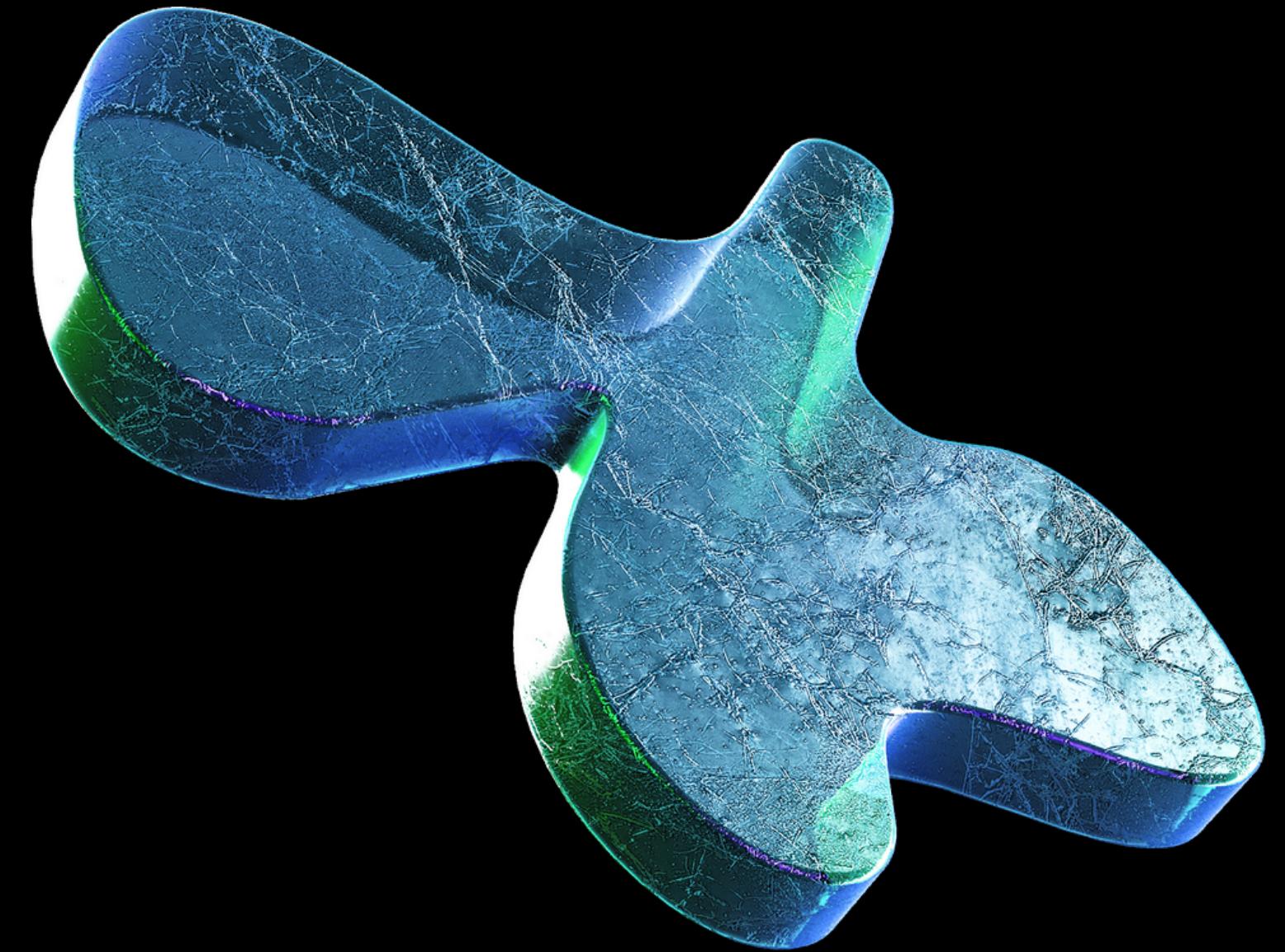
## Ejemplo: Llamado a una función como parte de una expresión

```
IF sal_ok (new_sal, level) THEN
  ...
END IF;
```

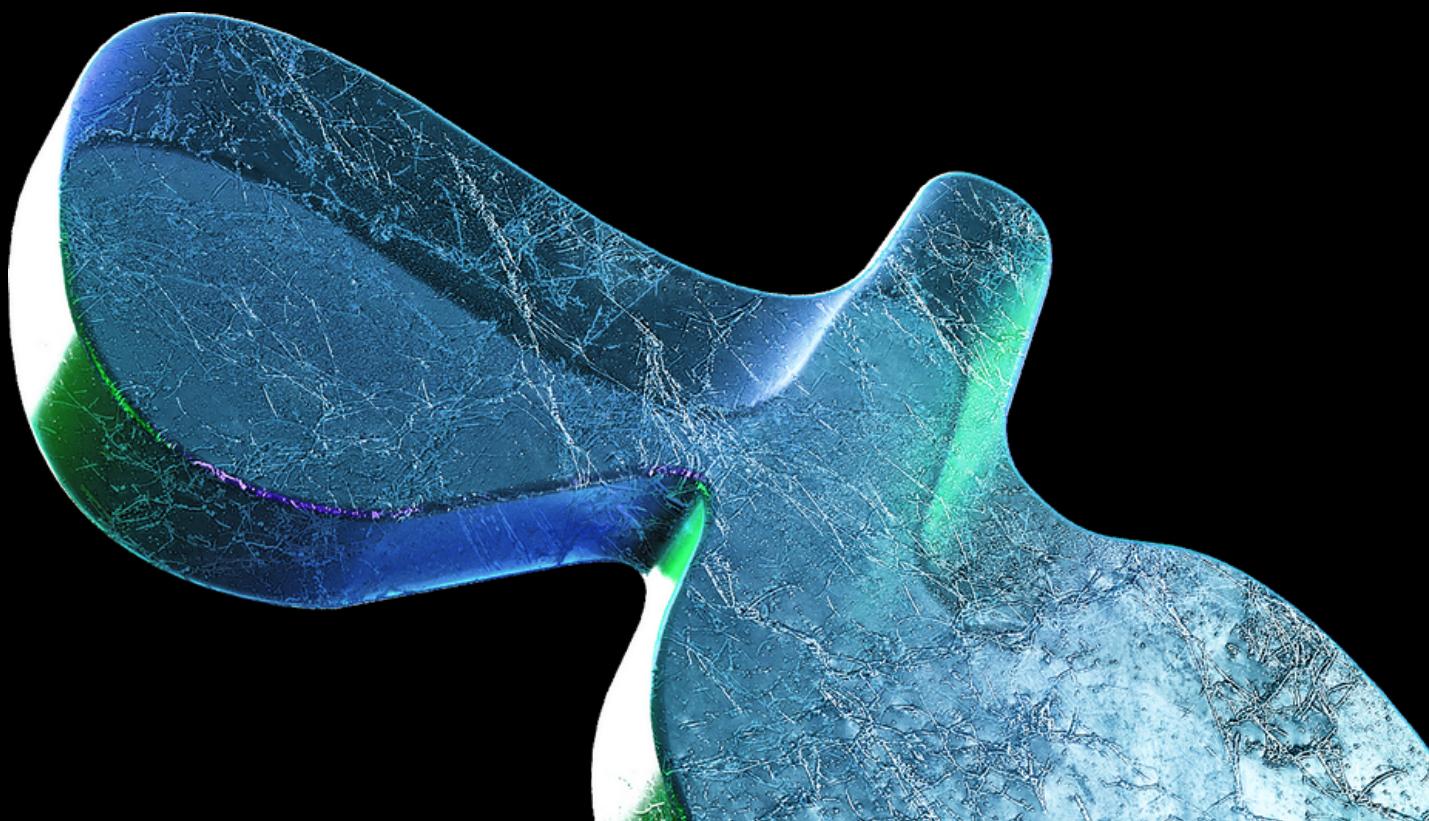
ó

```
promotable := sal_ok (new_sal, level) AND (rating > 3);
```

# Procedimientos y Funciones Almacenadas



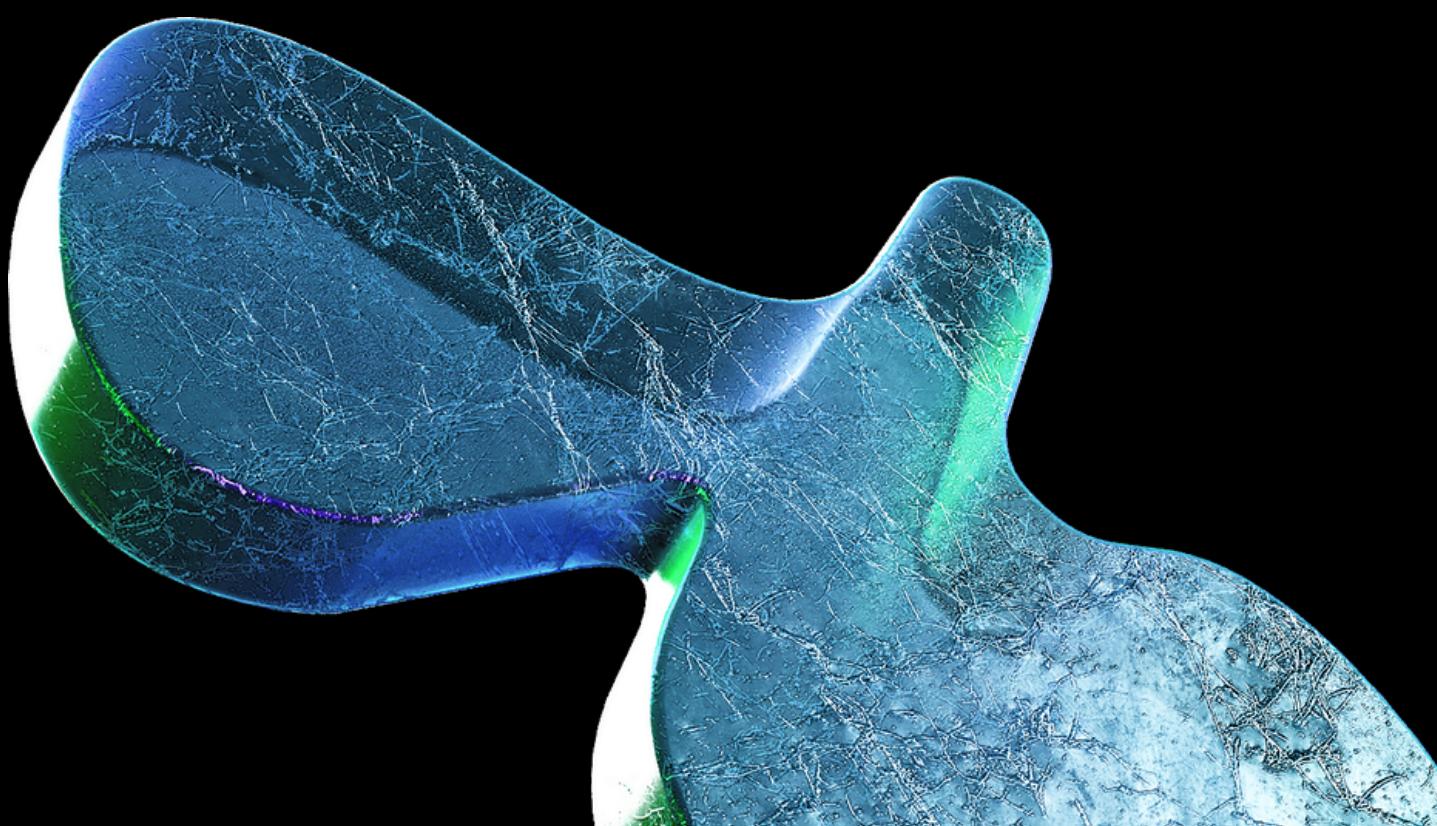
# Procedimientos y Funciones Almacenadas



Ejemplo:

```
CREATE FUNCTION get_bal(acc_no IN NUMBER)
    RETURN NUMBER IS
        acc_bal  NUMBER(11,2);
BEGIN
    SELECT balance INTO acc_bal
        FROM accounts
        WHERE account_id = acc_no;
    RETURN(acc_bal);
END;
```

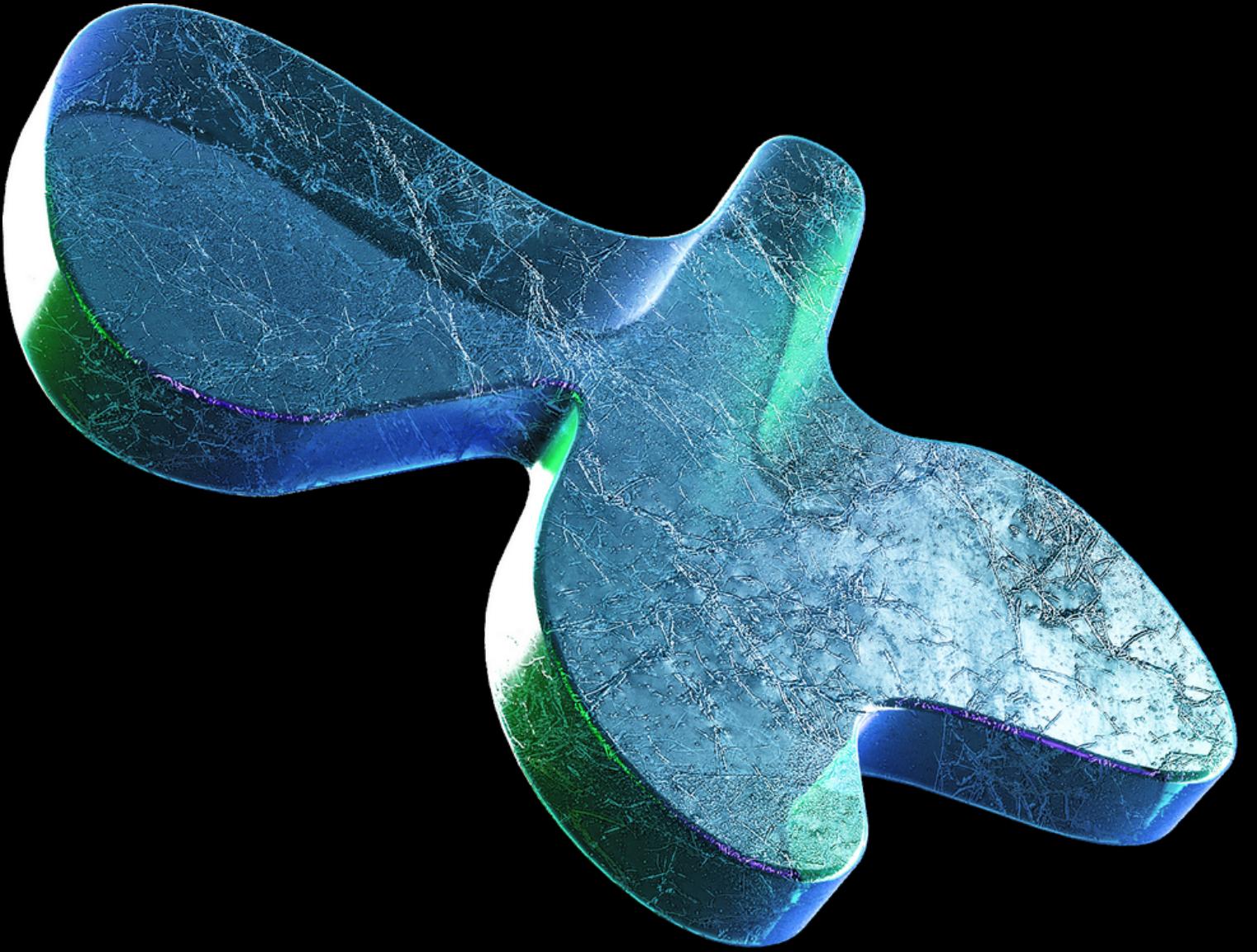
# Procedimientos y Funciones Almacenadas



Ejemplo:

```
CREATE PROCEDURE sam.credit (acc_no IN NUMBER,  
                             amount IN NUMBER) AS  
BEGIN  
    UPDATE accounts  
        SET balance = balance + amount  
        WHERE account_id = acc_no;  
END;
```

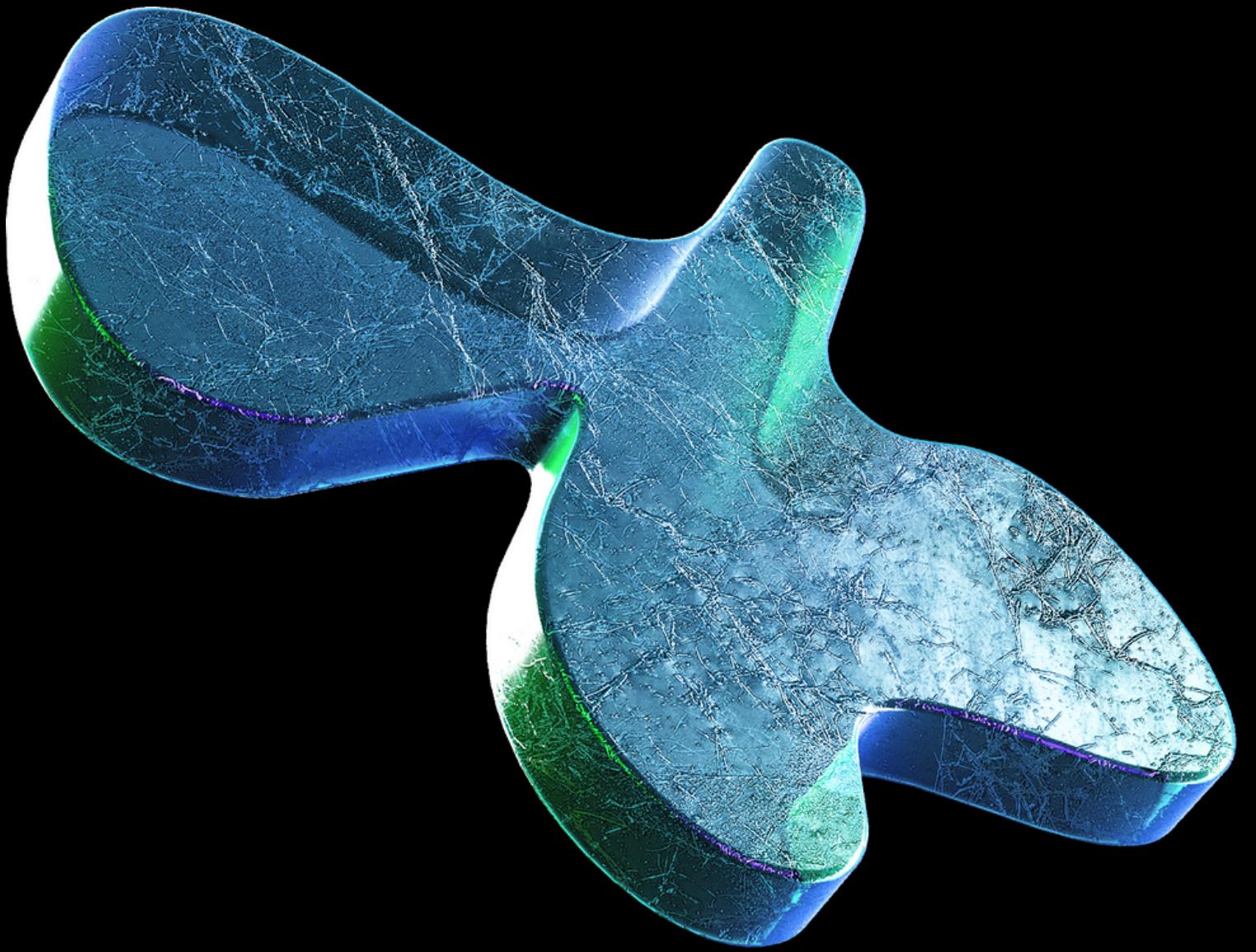
# Locales Vs Almacenados



Si se desarrolla un subprograma útil, entonces se tendrá una alta posibilidad de que éste sea llamado por más de un bloque. De acuerdo a esto, el subprograma debería ser almacenado en la base de datos.

Los únicos procedimientos y funciones que podrían declararse locales a un bloque, deberían tender a ser cortos, y sólo ser llamados por una sección específica del programa (el bloque que los contiene).

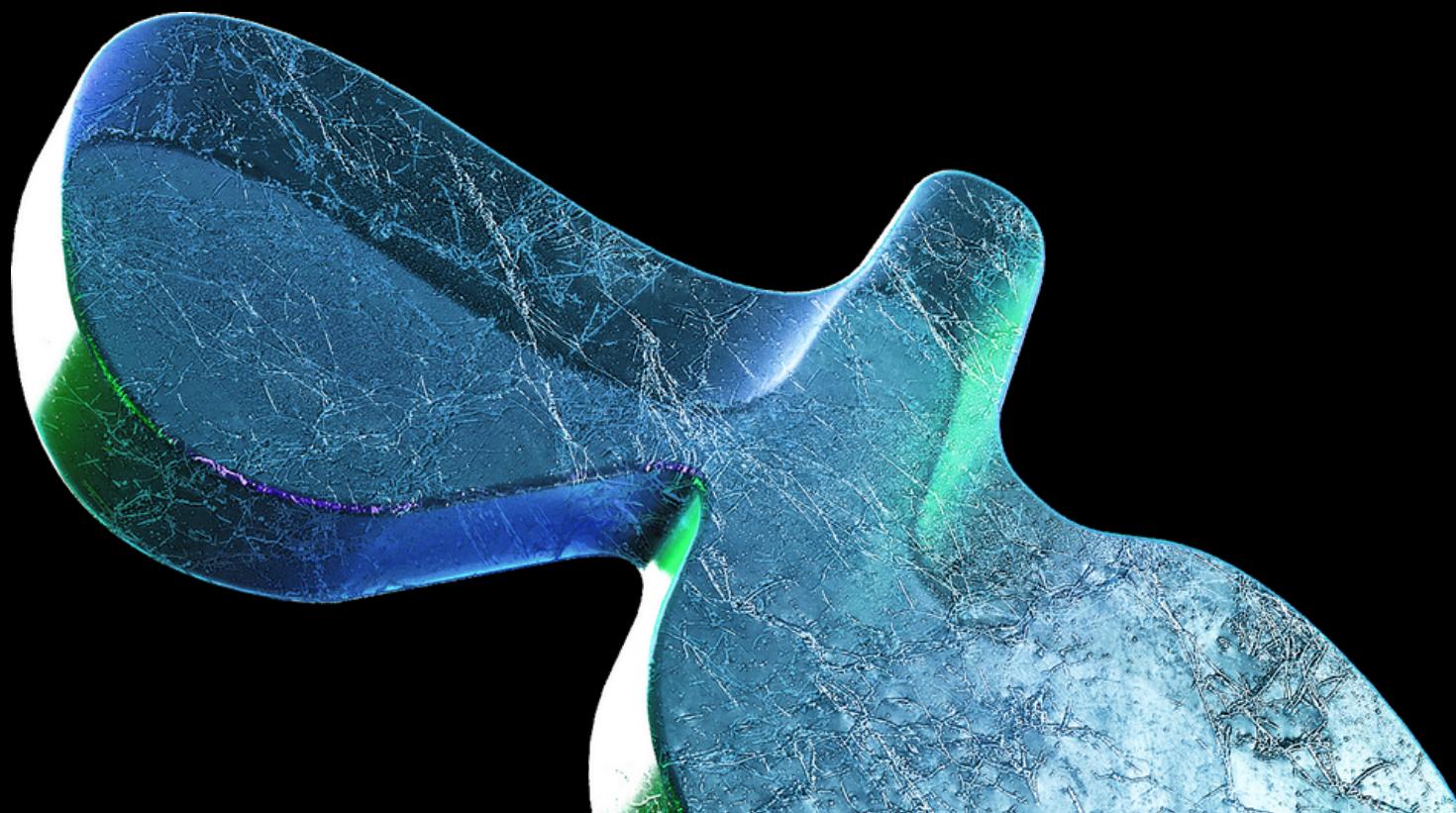
# Paquetes



Un *paquete* es una colección encapsulada de:

- Procedimientos
- Funciones
- Variables
- Constantes
- Cursosres
- Excepciones

# Paquetes



## A. Formato General (Pseudocódigo):

```
PACKAGE <nombre_paquete> IS
/*
 ** ESPECIFICACION DEL PAQUETE
 ** Declaraciones públicas de tipos de datos, variables,
 ** constantes, excepciones y cursores.
 ** Especificaciones públicas de procedimientos y funciones
 ** (declaraciones “forward”)
 */
END [nombre_paquete];
```

```
PACKAGE BODY <nombre_package> IS
/*
 ** CUERPO DEL PAQUETE
 ** Declaraciones privadas de tipos de datos, variables,
 ** constantes excepciones y cursores.
 ** Cuerpos de procedimientos y funciones públicos y privados.
 */
```

[BEGIN

-- sentencias de inicialización]

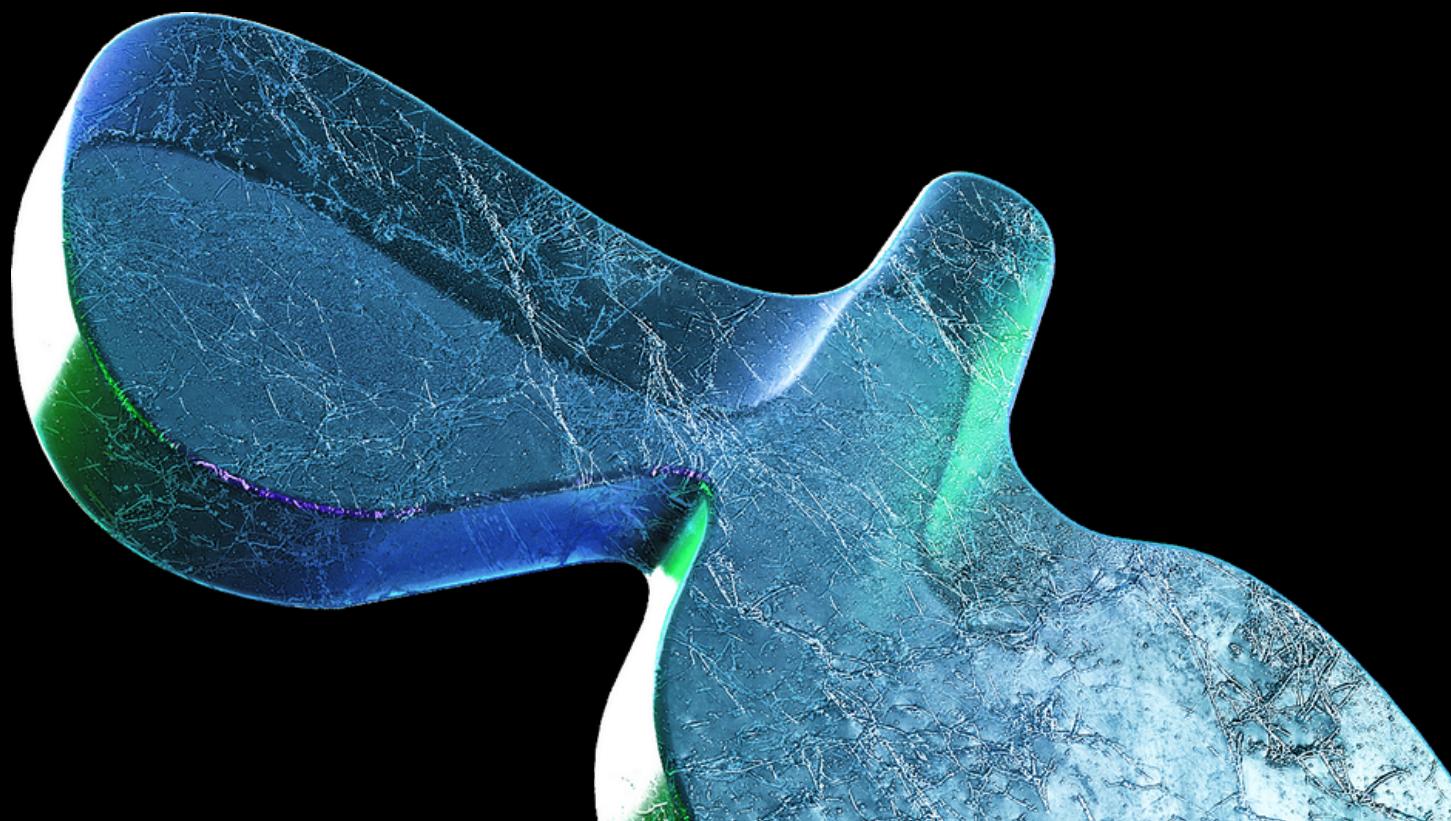
END [nombre\_paquete];

*Se ejecutan una sola vez, cuando se invoca el primer procedimiento o función*

donde:

**nombre\_paquete**, es un nombre único dentro del alcance de un esquema de base de datos

# Paquetes



Ejemplo:

```
CREATE PACKAGE emp_mgmt AS  
    FUNCTION hire (ename VARCHAR2, job VARCHAR2,  
                   mgr NUMBER, sal NUMBER, comm NUMBER,  
                   deptno NUMBER) RETURN NUMBER;
```

*Recuerde que el tipo de los parámetros, si corresponden, debe ser del mismo tipo que el atributo correspondiente en la tabla.  
Por ejemplo: ename emp.ename%type;*

```
    FUNCTION create_dept (dname VARCHAR2,  
                          loc VARCHAR2) RETURN NUMBER;
```

```
    PROCEDURE remove_emp (empno NUMBER);
```

```
    PROCEDURE remove_dept (deptno NUMBER);
```

```
    PROCEDURE increase_sal (empno NUMBER,  
                            sal_incr NUMBER);
```

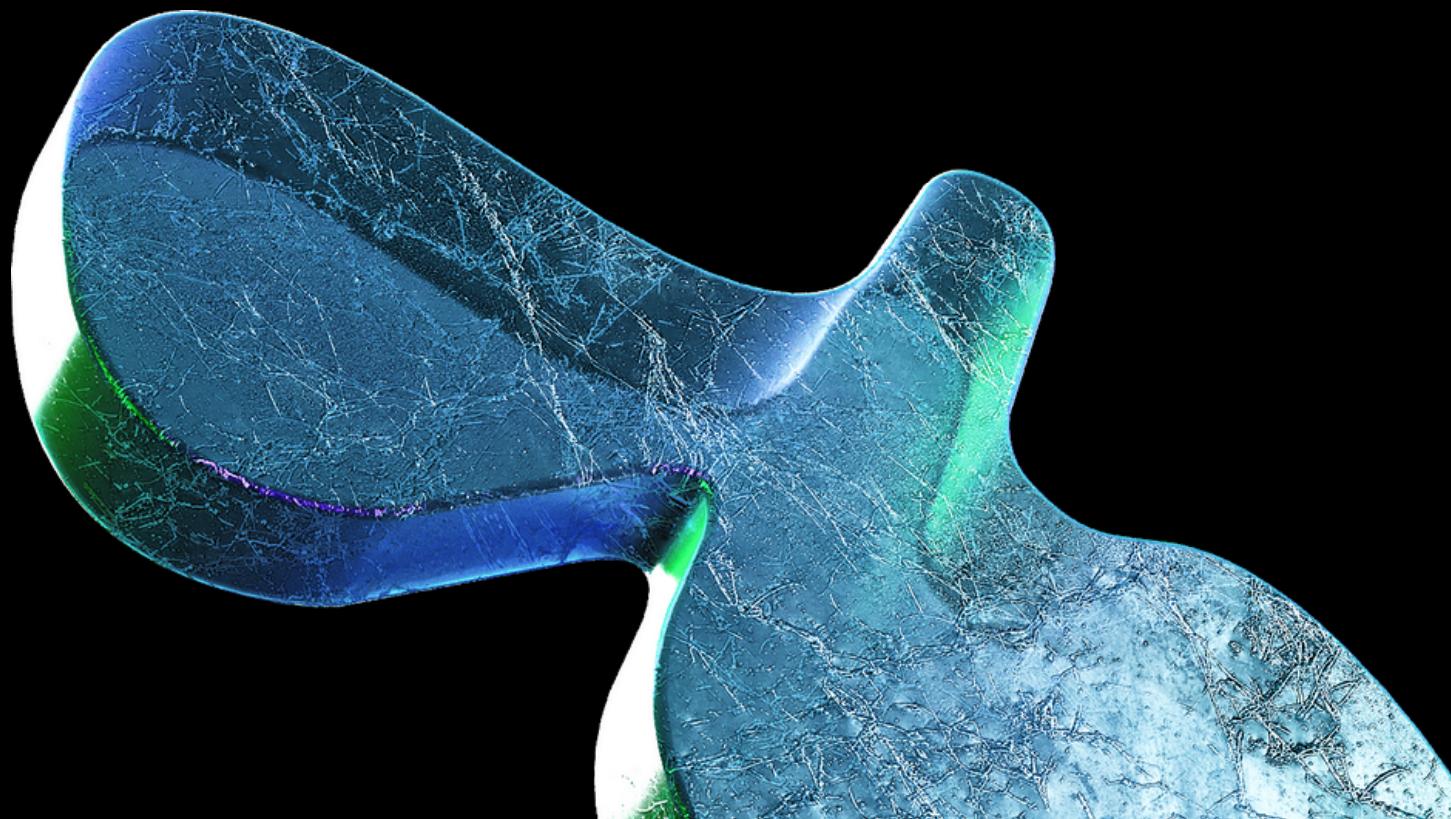
```
    PROCEDURE increase_comm (empno NUMBER,  
                            comm_incr NUMBER);
```

```
        no_comm      EXCEPTION;
```

```
        no_sal       EXCEPTION;
```

```
END emp_mgmt
```

# Paquetes



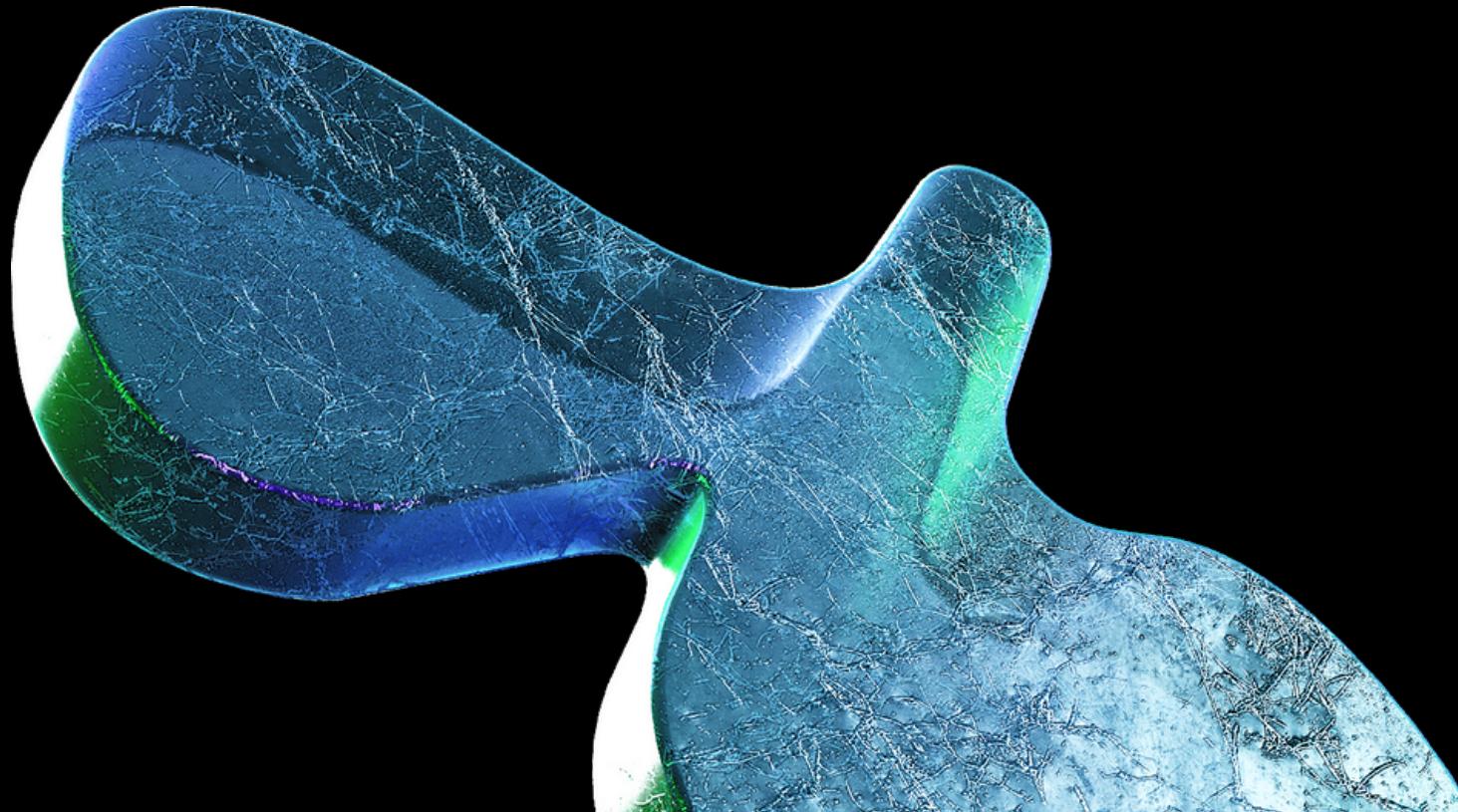
**Ejemplo:**

```
CREATE PACKAGE BODY emp_mgmt AS
    tot_emps NUMBER;
    tot_depts NUMBER;

    FUNCTION hire (ename VARCHAR2, job VARCHAR2,
                   mgr NUMBER, sal NUMBER, deptno NUMBER)
        RETURN NUMBER IS
            new_empno NUMBER(4);

    BEGIN
        SELECT empseq.NEXTVAL
              INTO new_empno
              FROM DUAL;
        INSERT INTO emp
              VALUES (new_empno, ename, job, mgr,
                      sal, comm, deptno)
        tot_emps := tot_emps + 1;
        RETURN (new_empno);
    END;
```

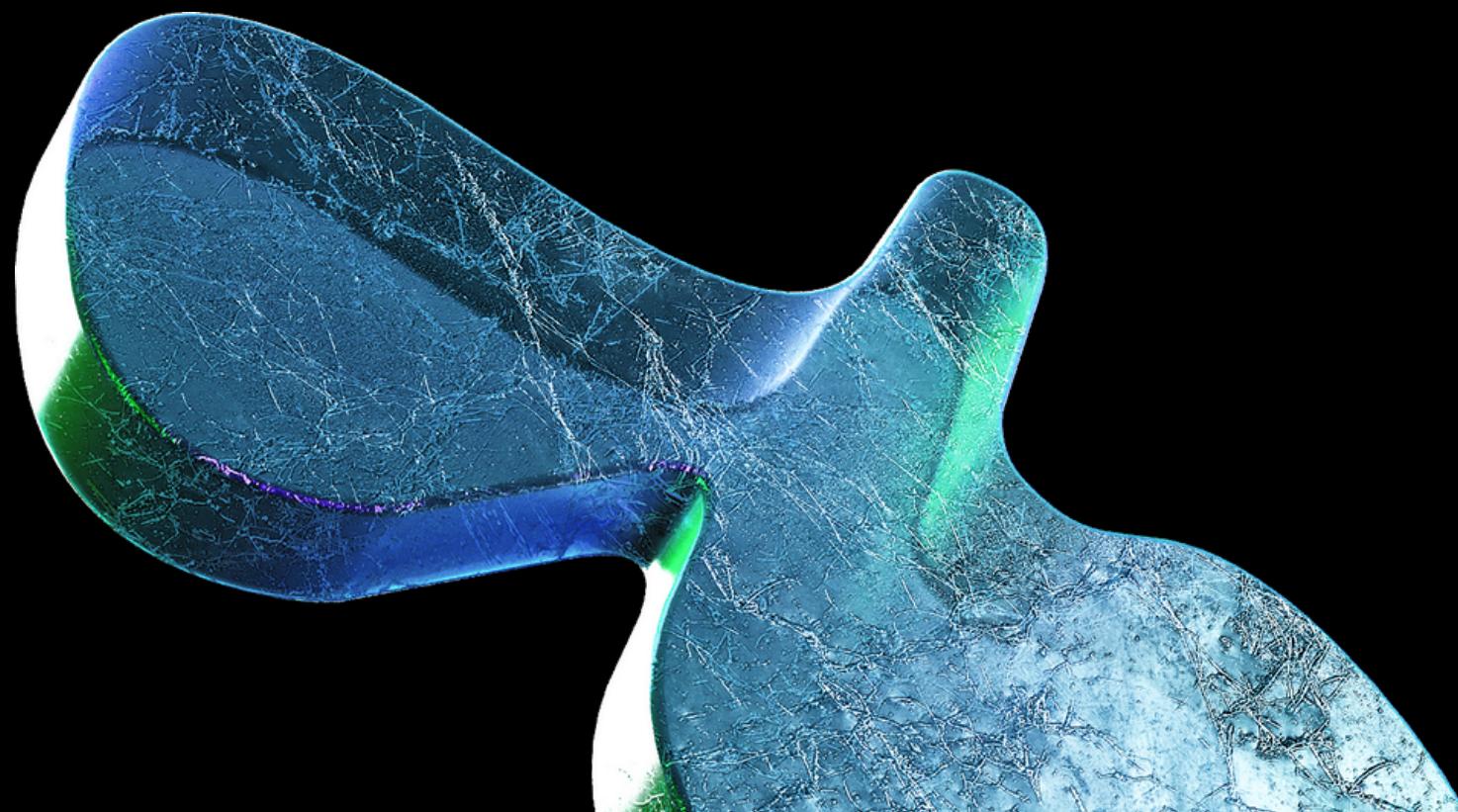
# Paquetes



```
FUNCTION create_dept(dname VARCHAR2, loc VARCHAR2)
    RETURN NUMBER IS
        new_deptno NUMBER(4);
BEGIN
    SELECT deptseq.NEXTVAL
        INTO new_deptno
        FROM dual;
    INSERT INTO dept
        VALUES (new_deptno, dname, loc);
    tot_depts := tot_depts + 1;
    RETURN (new_deptno);
END;

PROCEDURE remove_emp(empno NUMBER) IS
BEGIN
    DELETE FROM emp
        WHERE emp.empno = remove_emp.empno;
    tot_emps := tot_emps - 1;
END;
```

# Paquetes



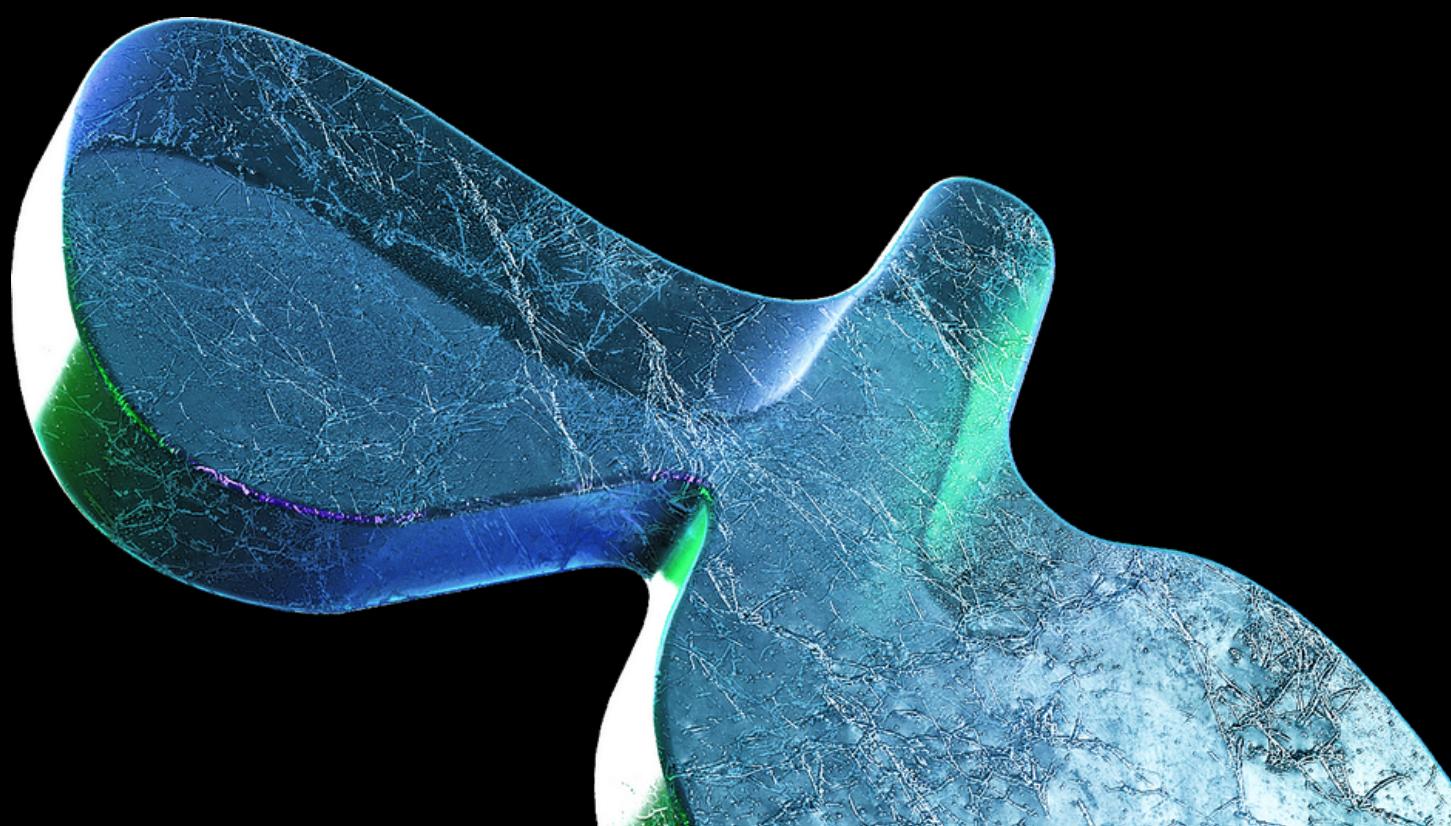
```
PROCEDURE remove_dept(deptno NUMBER) IS
BEGIN
    DELETE FROM dept
        WHERE dept.deptno = remove_dept.deptno;
    tot_depts := tot_depts - 1;
    SELECT COUNT(*) INTO tot_emps FROM emp;

/* En caso de que ORACLE elimine empleados desde la
   tabla EMP, para asegurar las restricciones de integridad
   referencial, se resetea el valor de la variable TOT_EMPS
   al número total de empleados en la tabla EMP.*/
END;
```

```
PROCEDURE increase_sal (empno NUMBER,
                           sal_incr NUMBER) IS
    curr_sal NUMBER(7,2);
```

```
BEGIN
    SELECT sal INTO curr_sal FROM emp
        WHERE emp.empno = increase_sal.empno;
    IF curr_sal IS NULL THEN
        RAISE no_sal;
    ELSE
        UPDATE emp SET sal = sal + sal_incr
            WHERE emp.empno = increase_sal.empno;
    END IF;
END;
```

# Paquetes



```
PROCEDURE increase_comm (empno NUMBER,  
                        comm_incr NUMBER) IS  
    curr_comm NUMBER(7,2);
```

```
BEGIN
```

```
    SELECT comm INTO curr_comm FROM emp  
        WHERE emp.empno = increase_comm.empno;
```

```
    IF curr_comm IS NULL THEN
```

```
        RAISE no_comm;
```

```
    ELSE
```

```
        UPDATE emp SET comm = comm + comm_incr  
            WHERE emp.empno = increase_comm.empno;
```

```
    END IF;
```

```
END;
```

```
...
```

```
BEGIN
```

```
-- Aquí va la inicialización de las  
-- variables tot_emps y tot_depts
```

```
END
```

*Esto se ejecuta una sola vez, cuando se invoca el primer subprograma del paquete*

```
EXCEPTION
```

```
when no_sal
```

```
---
```

```
---
```

```
when no_comm
```

```
---
```

```
---
```

```
END emp_mgmt;
```

# Paquetes

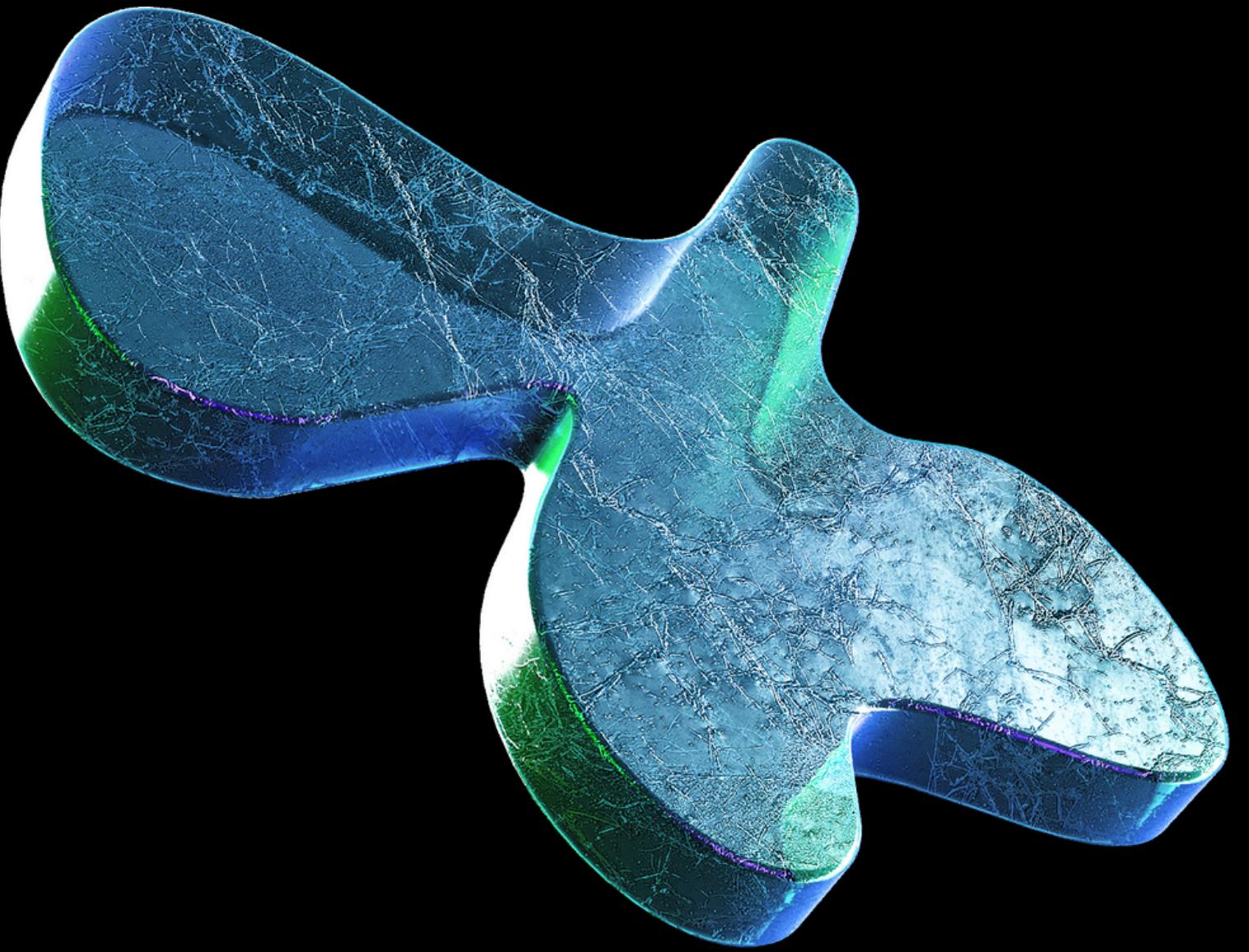
## Referencias al contenido del paquete

Los contenidos públicos del paquete pueden ser referenciados, usando notación de punto.

Ejemplo:

- nombre\_paquete.nombre\_tipo\_dato
  -
- nombre\_paquete.nombre\_variable
  -
- nombre\_paquete.nombre\_procedimiento(...);
  -
- nombre\_paquete.nombre\_funcion(...);

# Triggers

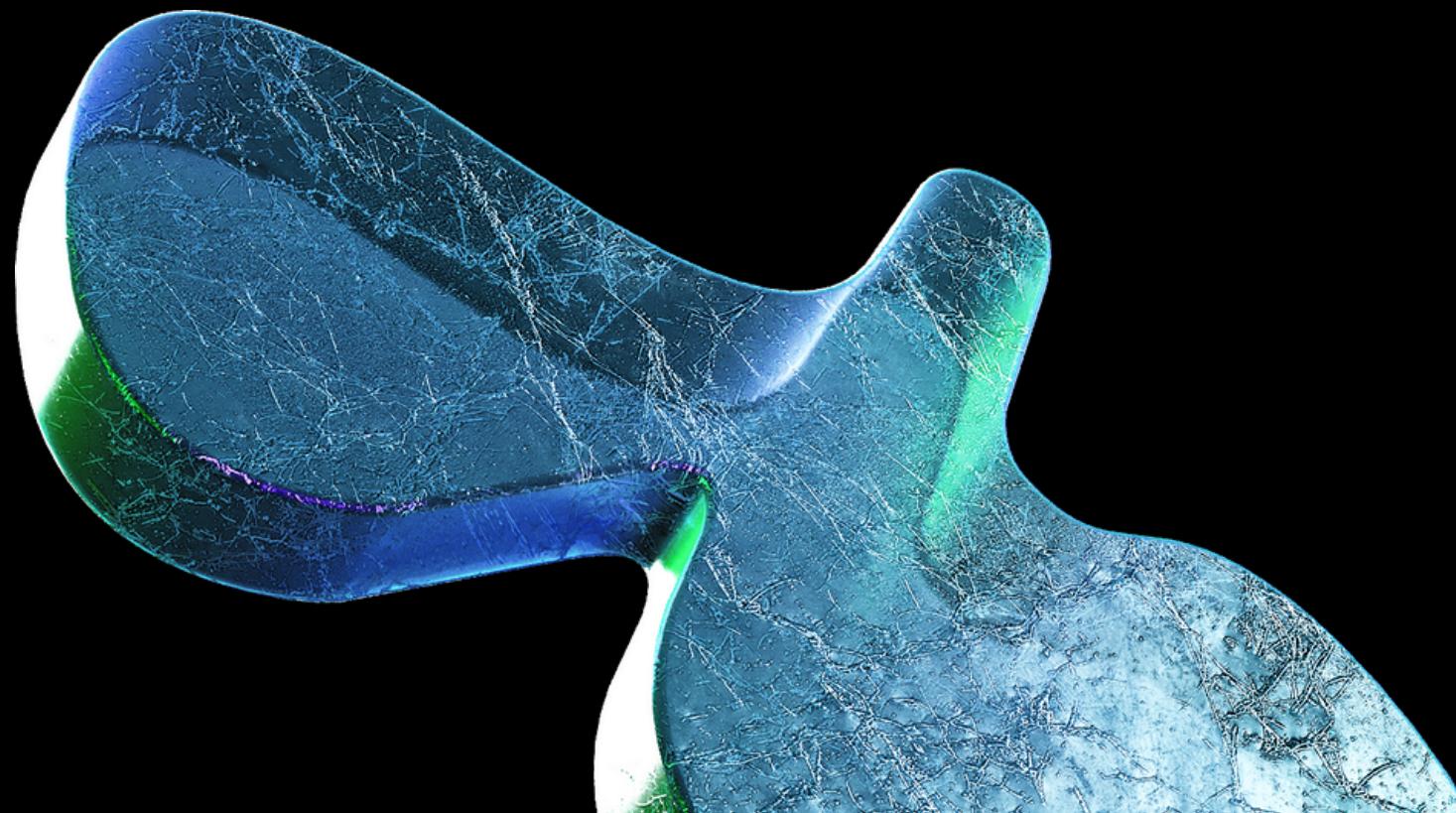


Es un procedimiento que el sistema ejecuta automáticamente, como un efecto secundario de una modificación de la base de datos.

Para diseñar un mecanismo de disparador se deben:

- Especificar las condiciones bajo las cuales se va a ejecutar el disparador.
- Especificar las acciones que se van a tomar cuando se ejecute el disparador.

# Triggers



Los triggers son usados a menudo, para iniciar procesos de negocio secundarios. Por ejemplo, pueden ser usados para comenzar las siguientes operaciones:

- Verificar la integridad de los datos en inserciones o actualizaciones.
- Implementar eliminaciones en cascada.
- Transparentemente efectuar eventos de bitácora (log).
- Asegurar el cumplimiento de complejas reglas de negocios.
- Iniciar procesos de negocios.
- Derivar valores de columnas automáticamente.
- Forzar complejas reglas de seguridad.
- Mantención de datos replicados.

## SINTAXIS:

```
CREATE [OR REPLACE] TRIGGER <nombre_trigger>
    {BEFORE | AFTER}
    {DELETE | INSERT | UPDATE [OF nombre_columna1
        [, nombre_columna2] ... ] }
    [OR {DELETE | INSERT | UPDATE [OF columnna
        [, columnna] ... ]}] ...
    ON <nombre_tabla>
```

```
[ [ REFERENCING {OLD [AS] old [NEW [AS] new]
    | NEW [AS] new [OLD [AS] old] } ]
```

```
[FOR EACH ROW
```

```
[WHEN (condición) ] ]
```

```
[DECLARE]
```

```
...
```

```
BEGIN
```

```
...
```

```
[EXCEPTION
```

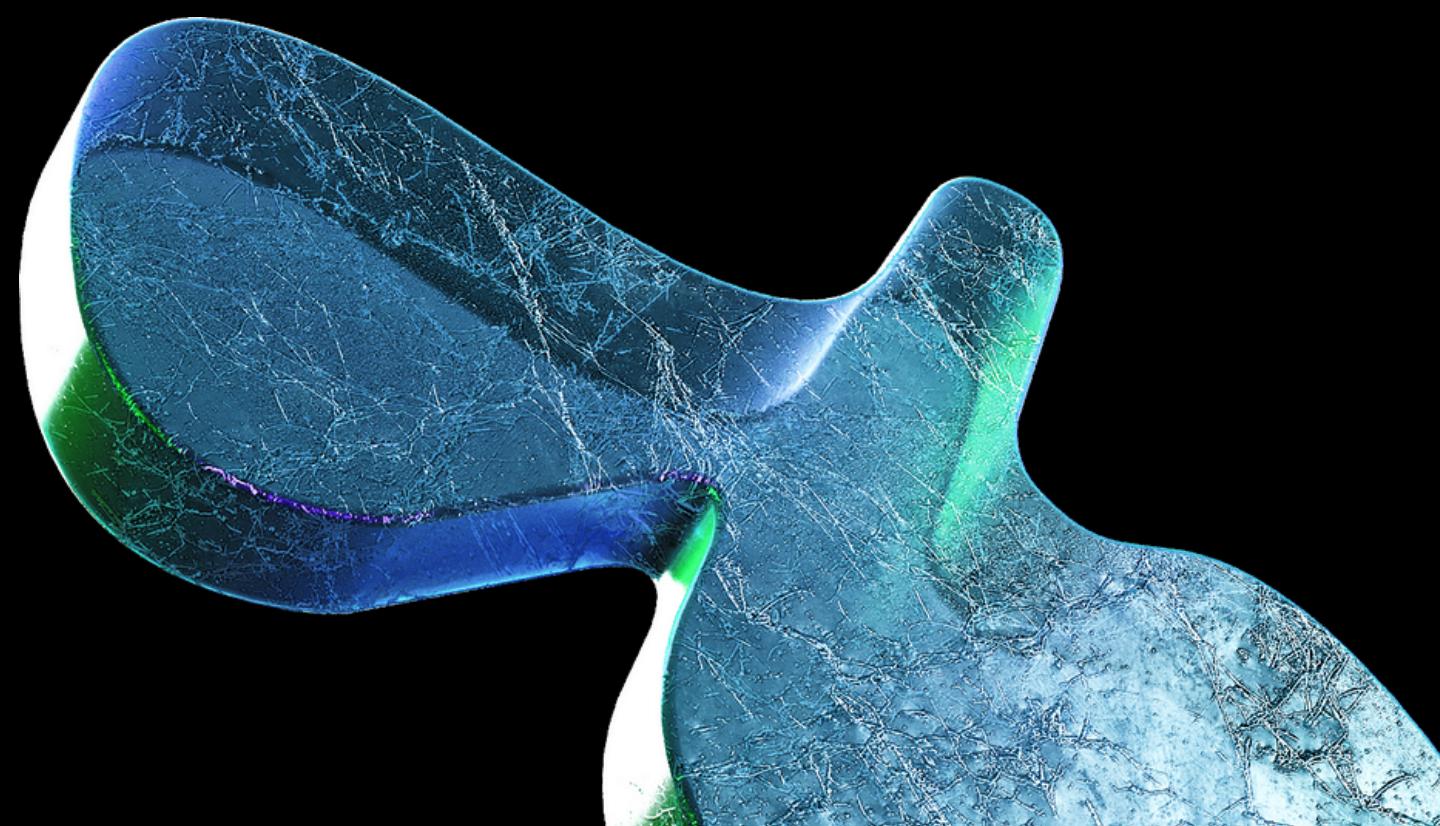
```
... ]
```

```
END;
```

*Cuerpo del trigger*

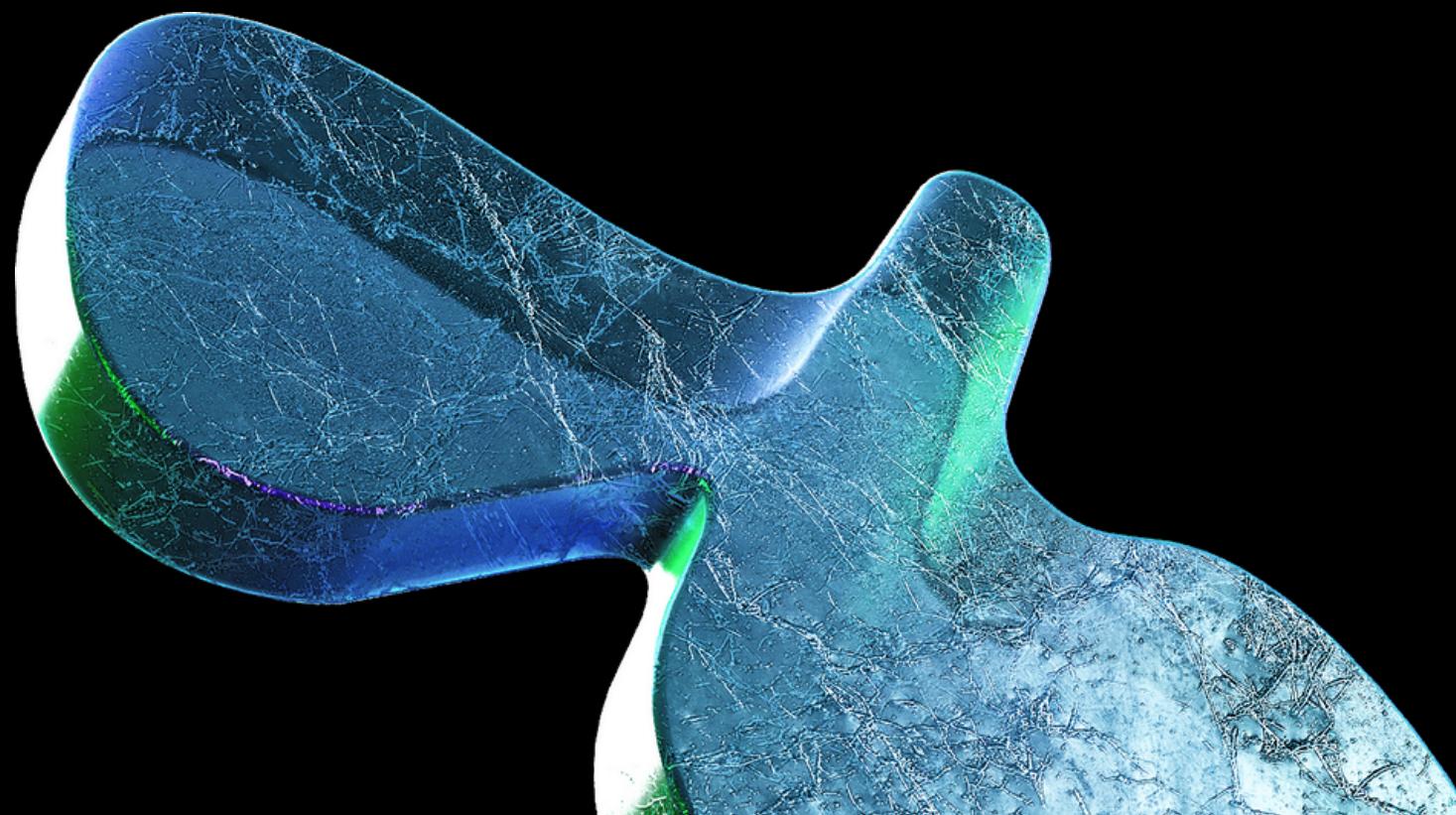
*En el cuerpo del trigger **no** puede haber:*

- Una instrucción SQL que implique la tabla por la cual se está gatillando el trigger
- Un commit
- Un rollback



# Triggers

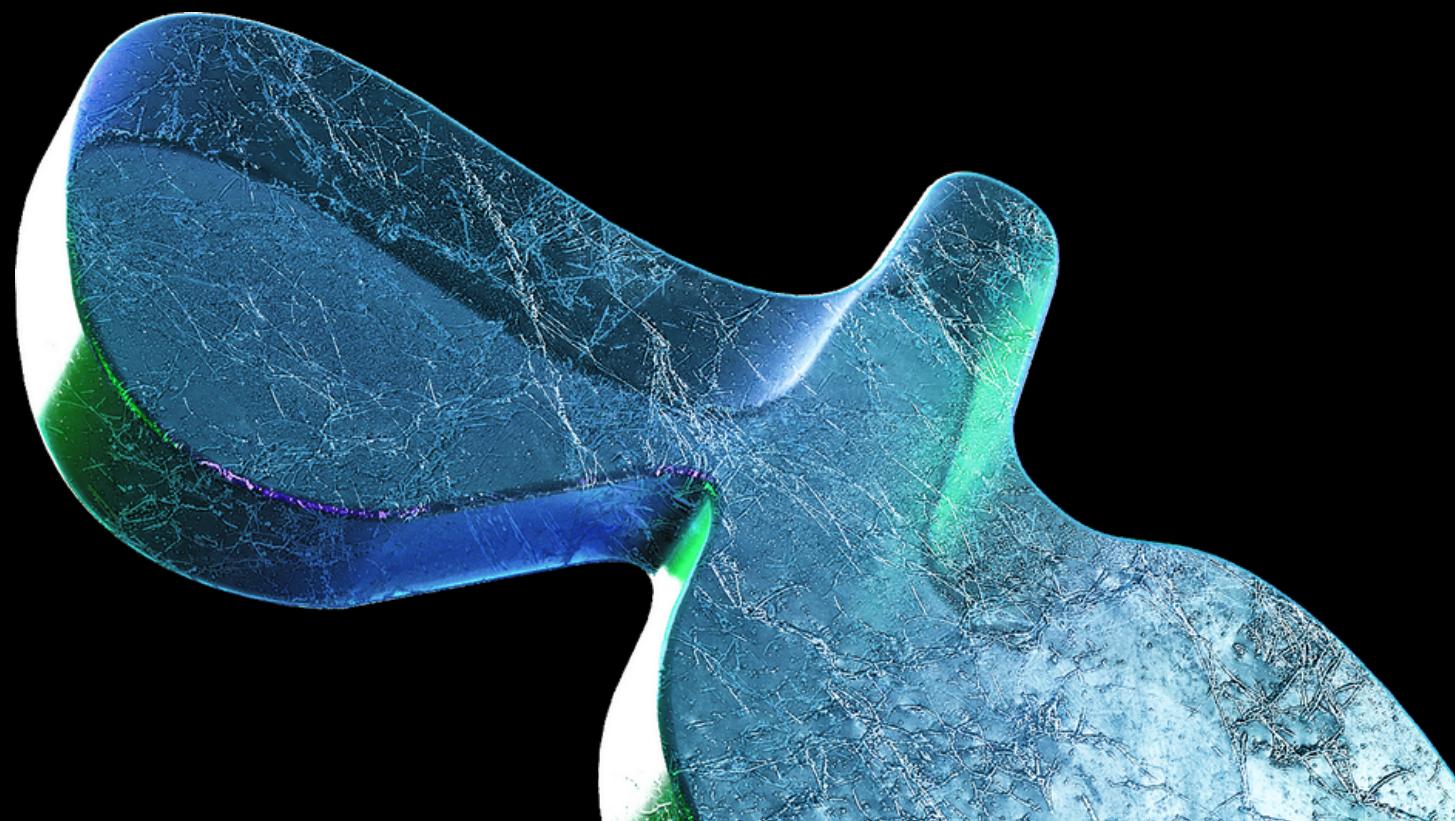
# Triggers



Cada trigger tiene tres opciones:

- La primera opción, que considera 2 casos, especifica cuando es iniciado el trigger: BEFORE (antes) ó AFTER (después) de una operación específica sobre la tabla.
- La segunda opción, que considera 3 casos, especifica la operación que activa el trigger: INSERT, UPDATE o DELETE. Estas operaciones pueden ser especificadas individualmente, o en cualquier combinación incluyéndolas todas a la vez. Por ejemplo:
  - INSERT
  - INSERT OR DELETE
  - DELETE OR UPDATE
  - INSERT OR UPDATE
  - DELETE OR UPDATE OR INSERT

# Triggers

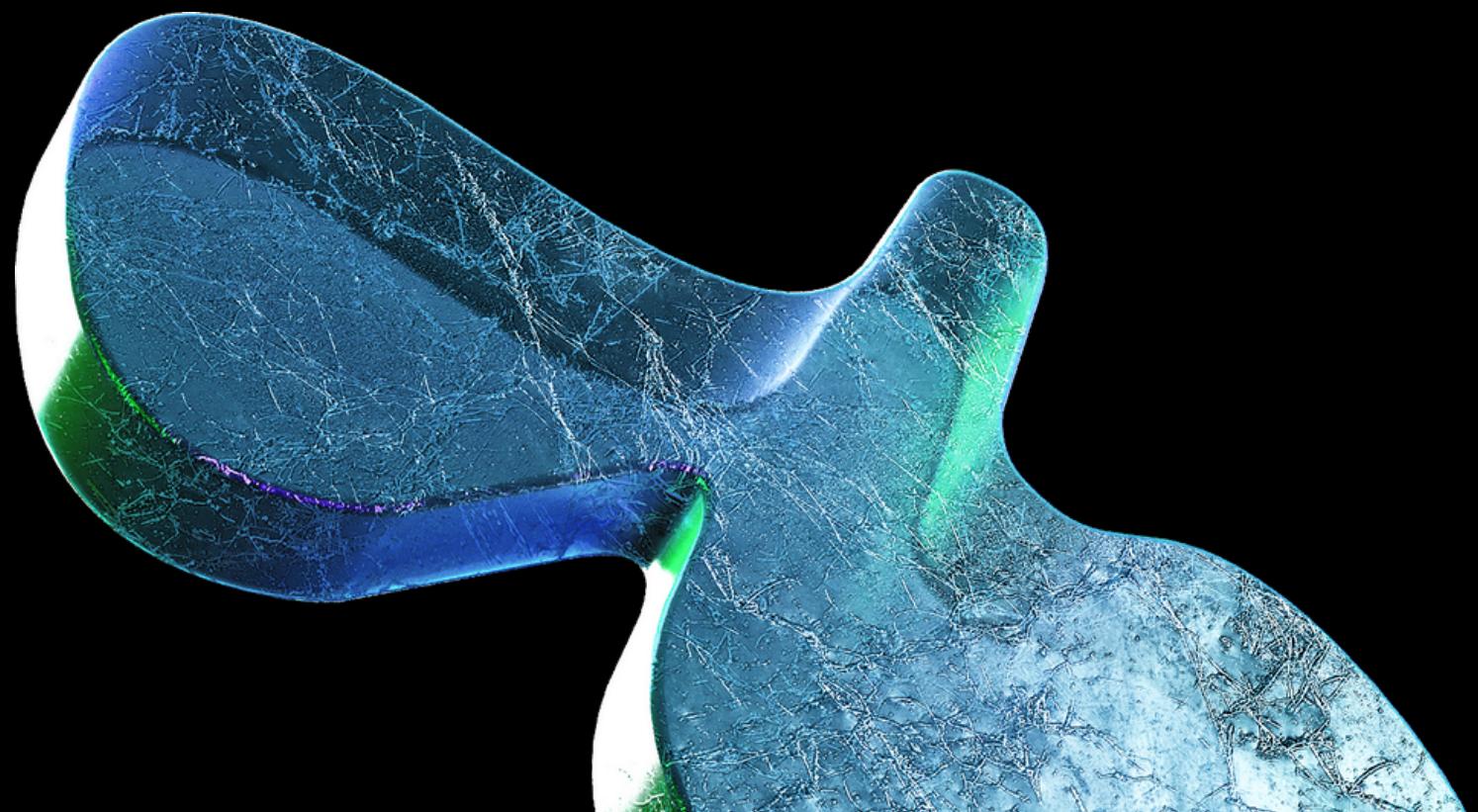


- La tercera opción, que considera 2 casos, especifica si el trigger será ejecutado sólo una vez por cada operación sobre la tabla (default), o una vez para cada fila afectada por la operación que activa el trigger (FOR EACH ROW). En este último caso, se tiene un trigger de fila.

## Opciones de los triggers de base de datos

<b>Before o After</b>	<b>Insert, Update o Delete</b>	<b>For Each Row</b>
BEFORE	INSERT	
BEFORE	UPDATE	
BEFORE	DELETE	
BEFORE	INSERT	FOR EACH ROW
BEFORE	UPDATE	FOR EACH ROW
BEFORE	DELETE	FOR EACH ROW
AFTER	INSERT	
AFTER	UPDATE	
AFTER	DELETE	
AFTER	INSERT	FOR EACH ROW
AFTER	UPDATE	FOR EACH ROW
AFTER	DELETE	FOR EACH ROW

# Triggers



## Pseudoregistros :old y :new

Un trigger a nivel de fila se dispara una vez por cada fila procesada. Dentro del trigger, se puede accesar la fila que está siendo actualmente procesada, a través de los pseudoregistros: `:old` y `:new`

El tipo de ambos es:

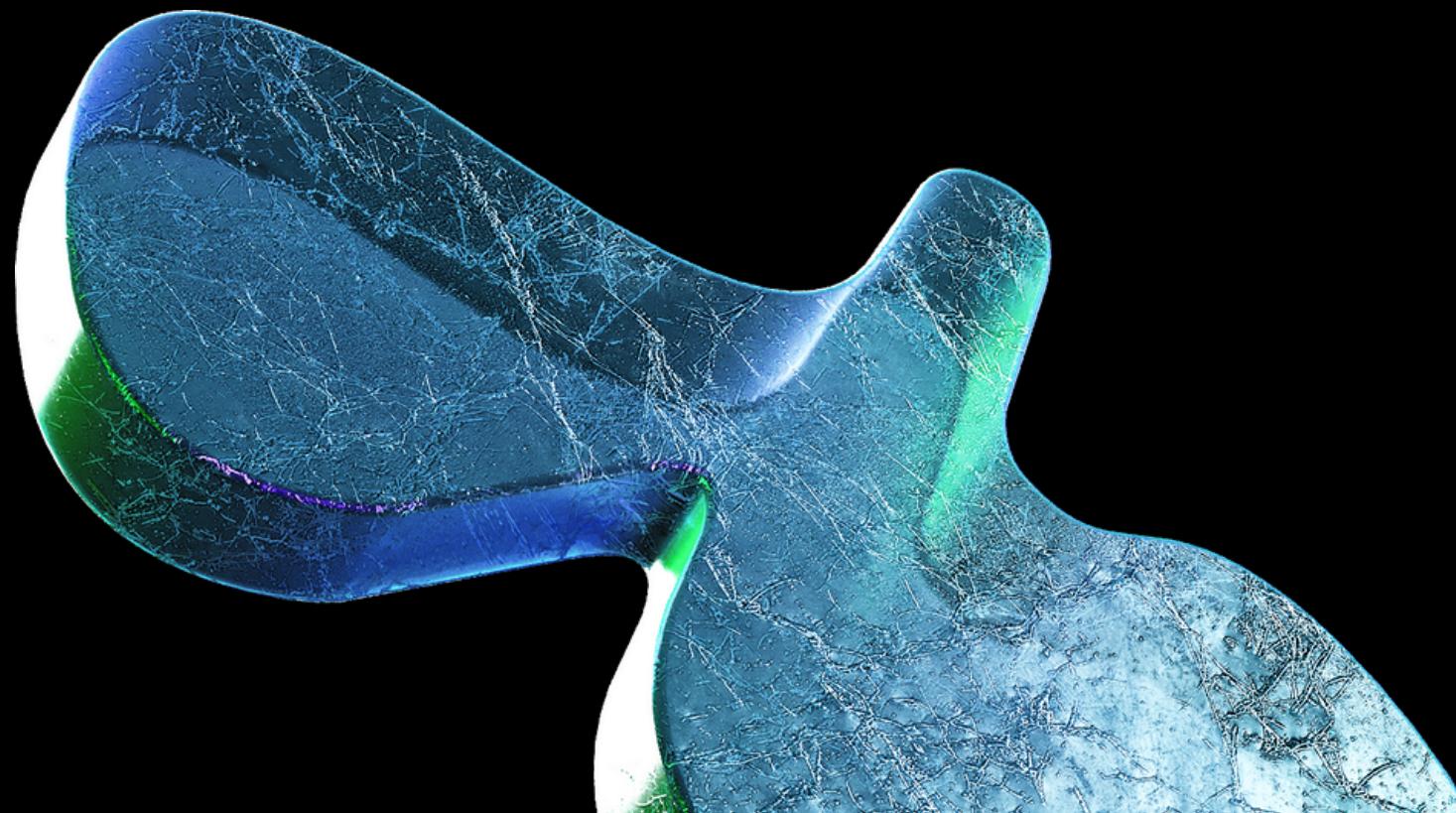
`triggering_table%ROWTYPE;`

donde `triggering_table` es la tabla para la cual se define el trigger.

## :old y :new

Sentencias del Trigger	:old	:new
INSERT	Indefinido. Todos los campos son nulos.	Valores que serán insertados cuando la sentencia está completa.
UPDATE	Valores originales para la fila antes de la actualización.	Nuevos valores que serán actualizado cuando la sentencia está completa.
DELETE	Valores originales antes de que la fila sea eliminada.	Indefinido. Todos los campos son nulos.

# Triggers



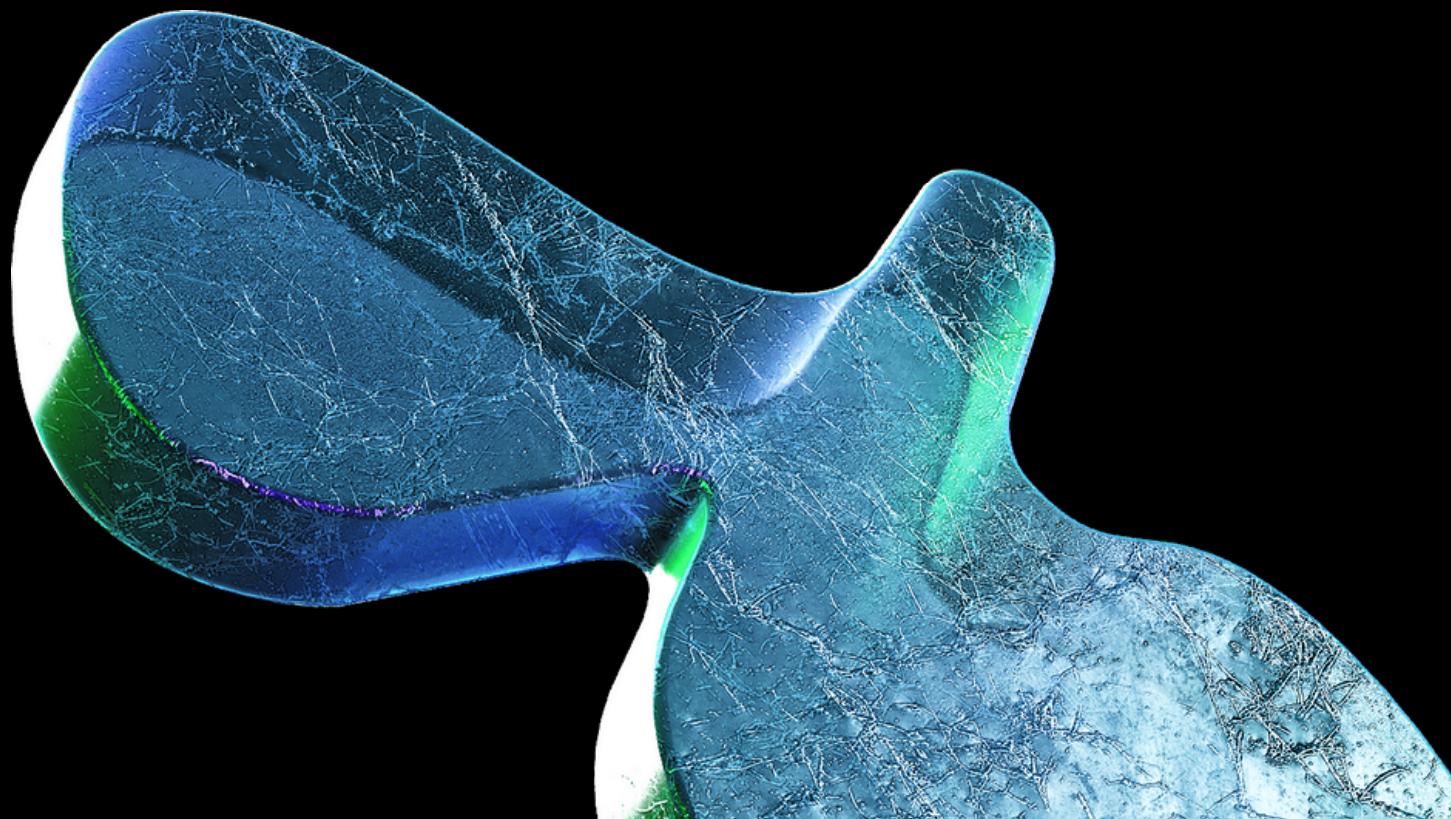
## Opciones condicionales

Debido a que un trigger de base de datos puede ser creado para múltiples operaciones de una tabla (por ejemplo, “INSERT OR DELETE OR UPDATE OF...”), las opciones condicionales o predicados pueden ser usados en el cuerpo del trigger para distinguir entre que operación ha causado el disparo del trigger.

Estos predicados condicionales son **INSERTING**, **DELETING** y **UPDATING**.

Ellos son usados para ejecutar secciones específicas del cuerpo de un trigger de base de datos, según la operación que disparó el trigger de base de datos.

# Triggers



```
CREATE TRIGGER scott.salary_check
  BEFORE
  INSERT OR UPDATE OF sal, job
  ON scott.emp
  FOR EACH ROW
  WHEN (new.job <> 'PRESIDENT')
  DECLARE
    minsal  NUMBER;
    maxsal  NUMBER;
  BEGIN
    /*Obtiene el salario mínimo y el máximo para el */
    /*trabajo del empleado desde la tabla SAL_GUIDE */
    SELECT minsal, maxsal
      INTO minsal, maxsal
      FROM sal_guide
      WHERE job = :new.job;
    /*Si el salario del empleado es menor que el mínimo
    o arriba del máximo para ese trabajo, se genera
    un error */
    IF (:new.sal < minsal OR :new.sal > maxsal) THEN
      raise_application_error (-20601, 'Salario'
        || :new.sal || 'fuera del rango para el trabajo'
        || :new.job || ' para el empleado' || :new.ename);
    END IF;
  END;
```

# PL-SQL

