SQL

Base de Datos – C108

Autores: Viviana Ortiz – Ignacio Bisso

SQL – Introducción

- Structured Query Language
- Es el lenguaje más universalmente usado para bases de datos relacionales
- Lenguaje declarativo de alto nivel
- Desarrollado por IBM (1974-1977)
- Se convirtió en un standard definido por :
  - ANSI (American National Standards Institute) e
  - ISO (International Standards Organization)
- El standard actual es el SQL:1999 (aunque muchas DBMS no lo implementaron por completo aún. Existen revisiones del 2003 y 2006)

SQL - Introducción

Las Sentencias del SQL se dividen en:

- Sentencias DDL (Data Definition Language): Permiten crear/modificar/borrar estructuras de datos.
- Sentencias DML (Data Manipulation Languaje): para manipular datos
- Tambien provee sentencias para:
  - Definir permisos (control de acceso de usuarios)
  - Manejo de transacciones
  - Otros

SQL - Introducción

- Términos
  - tabla → relación
  - fila → tupla
  - columna → atributo

DDL - Create table

```
CREATE TABLE empleados (
  enombre    char(15) NOT NULL,
  ecod       integer  NOT NULL,
  efnac      date,
  dcod       integer
  )
```

Crea la tabla empleados con 4 columnas. La tabla no tendrá ninguna fila, hasta que no se ejecute un insert.

DDL - Create table

```
CREATE TABLE empleados (
  enombre    char(15) NOT NULL,
  ecod       integer  NOT NULL,
  efnac      date,
  dcod       integer
  ) Primary Key (edoc)
```

Es posible definir una clave primaria

DDL - Create table

```
CREATE TABLE empleados    (
  enombre     char(15) NOT NULL,
  ecod        integer  NOT NULL,
  efnac       date,
  dcod        integer
  ) Primary Key (edoc)
    Foreign Key dcod References
    Deptos
```

Define la columna dcod como clave foránea apuntando a Deptos

DDL – Sentencia Drop table

- **DROP TABLE** ***table;***

Ejemplo:

DROP TABLE empleados;

Borra la tabla y todas sus filas

DDL – Alter table

- Permite:
  - agregar columnas
  - cambiar la definición de columnas
  - agregar o borrar constraints

```
ALTER TABLE table
  ADD (column datatype [DEFAULT expr]);

ALTER TABLE table
  MODIFY (column datatype [DEFAULT expr] );

ALTER TABLE table
  ADD FOREIGN KEY (column [,...]),]
  REFERENCES table((column [,...]);
```

SQL - Instrucciones DML

Instrucciones DML: Permiten Manipular (leer y modificar) los datos almacenados en las tablas.

- INSERT: Crear nuevas filas en una tabla
- SELECT: Leer filas (o columnas) de tablas.
- UPDATE: Modificar filas existentes en una tabla
- DELETE: Borrar filas de una tabla.

DML - INSERT

- INSERT:
  - Agrega filas en una tabla.
  - **Unica** sentencia que provee SQL para agregar filas.
  - Existen 2 Formas de ejecutar el insert

1) **Usando la cláusula VALUES (agrega una sola fila por cada comando insert)**

```
INSERT INTO table [(column [, column...])]
VALUES (value [, value...]);
```

```
INSERT INTO empleados VALUES (1, 'Juan Perez', '04/04/98', 100)

INSERT INTO deptos (dcod, ddescr) VALUES (50, 'CONTABILIDAD')

INSERT INTO deptos VALUES (50, 'CONTABILIDAD')
```

2) **Usando la cláusula SELECT (agrega un conjunto de filas mediante un solo insert)**

Esta variante del INSERT La veremos más adelante……

DML – SELECT

SELECT [ALL/DISTINCT] *select_list*
FROM *table* [*table alias*] [,...]
[WHERE *condition*]
[GROUP BY *column_list*]
[HAVING *condition*]
[ORDER BY *column_name* [ASC/DESC] [,...]

DML – SELECT

SELECT a1, ..., an
FROM t1,....,tn
WHERE <cond>
ORDER BY ai, aj

En algebra relacional:

$$\Pi_{a1 \ldots an} (\delta_{<cond>}(t1 \,|X|\, .. \,|X|\, tn))$$

SELECT

SELECT ecod, enombre
FROM empleados
WHERE dcod=5;

En algebra relacional:

$$\Pi_{ecod,enombre} (\delta_{<dcod=5>}(empleados))$$

Obtener las columnas ecod, enombre de la tabla empleados de aquellas filas cuya columna dcod tiene el valor 5

SELECT (*)

- Para acceder a todas las columnas → *

```
SELECT *
FROM empleados
WHERE dcod=40
```

Obtener TODAS las columnas de la tabla empleados de aquellas filas cuya columna dcod tiene valor 40

DML - INSERT

- Volvamos al Insert:
  - También pueden insertarse un conjunto de filas

```
INSERT INTO table [(column [, column...])]
       SELECT...

INSERT INTO gerentes(gcod, gnombre, gsalario)
       SELECT ecod, enombre, esalario
       FROM   empleados
       WHERE  ecargo = 'GERENTE';
```

La cantidad de columnas y tipos que devuelve el select debe coincidir con la cantidad de columnas de la Tabla.

SELECT (Join)

SELECT enombre
FROM empleados, deptos
WHERE dcod = deptoid
AND dnombre = 'Sistemas'

Tabla Deptos	
deptoid	integer,
dnombre	char(30)
gerente	integer
pcod	ingteger

Condición de Junta

En algebra relacional:

$\Pi_{enombre}\ (\delta_{<dcod=deptoid\ AND\ dnombre='Sistemas'>}(empleados\ X\ deptos))$

Los empleados que trabajan en depto Sistemas

SELECT (join)

SELECT enombre, pnombre
FROM empleados, deptos, provincias
WHERE dcod = deptoid
AND pcod = provid
AND dnombre = 'Sistemas'

Tabla Empleados	
enombre	char(30),
ecod	integer,
Efnac	date,
dcod	integer

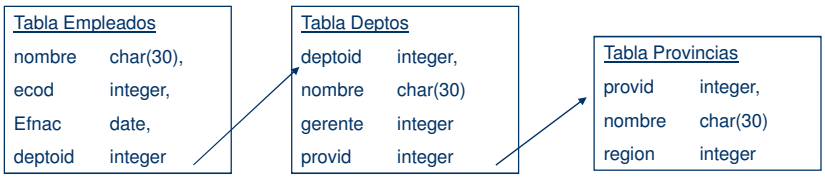
Tabla Deptos	
deptoid	integer,
dnombre	char(30)
gerente	integer
pcod	integer

Tabla Provincias	
provid	integer,
pnombre	char(30)
region	integer

SELECT (join)

Si los nombres de columnas se repiten, hay que anteponer el nombre de la tabla para evitar ambigüedades.

```
SELECT   empleados.nombre, provincias.nombre
FROM     empleados, deptos, provincias
WHERE    empleados.deptoid = deptos.deptoid
AND      deptos.provid = provincias.provid
AND      deptos.nombre = 'Sistemas'
```

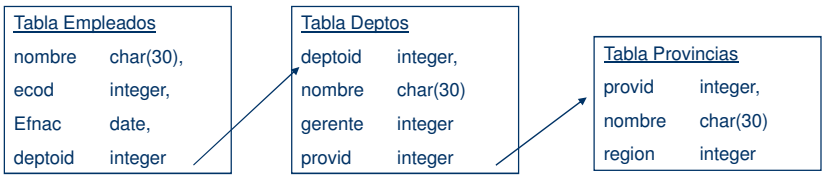


SELECT (Alias)

Puedo usar alias de tablas para simplificar el SQL.

```
SELECT   e.nombre, p.nombre
FROM     empleados e, deptos d, provincias p
WHERE    e.deptoid = d.deptoid
AND      d.provid = p.provid
AND      d.nombre = 'Sistemas'
```

Los Alias se usan mayormente para simplificar la escritura del SELECT, sin embargo algunos tipos de subqueries requieren el uso de alias, ya que de otra manera no es posible escribirlos



SELECT (outer join)

- Se incluyen las tuplas que cumplan la condición de join + aquellas en las cuales el valor de la columna que participa en el join, tiene valor nulo. En este caso todos los empleados, aunque no tengan departamento

SELECT enombre, ddescr

FROM empleados e LEFT OUTER JOIN departamento d ON d.dcod=e.dcod

Null Values

- En algunos casos no se dispone de un valor para asignar a una columna
  - *Por ejemplo: fecha de emisión del registro*
  - SQL provee un valor especial para estos casos: NULL

Null Values

Las columnas que no tienen ningún valor asignado contienen valor NULL

Ejemplo

```
create table T1 ( col1 integer, col2 integer, col3 integer)
insert into T1(col1, col3) values (9,9)   ⟶  El valor de col2 es NULL
insert into T1(col1, col2, col3) values (8,8,8)
```

```
db2 => select * from t1
C1          C2          C3
----------- ----------- -----------
          9           -           9
          8           8           8
2 registro(s) seleccionados.
```

```
db2=>select * from T1
        where C2 IS NULL
C1          C2          C3
----------- ----------- -----------
          9           -           9
1 registro(s) seleccionados.
```

```
db2 => select * from T1
        where C2 IS NOT NULL
C1          C2          C3
----------- ----------- -----------
          8           8           8
1 registro(s) seleccionados.
```

Null Values

- La presencia de *null* genera algunas complicaciones
  - Operador especial para controlar si un valor es nulo (IS NULL o IS NOT NULL).
  - "edad > 21" - true o false cuando edad es *null*? Qué pasa con el AND, OR y NOT ?
  - Surge la necesidad de una "3-valued logic" (true, false and *unknown*).
  - Hay que ser cuidadoso con la clausula WHERE.
  - En SQL el WHERE elimina toda fila que NO evalua a TRUE en el WHERE (O sea condiciones que evaluan a False o Unknown no califican.)

Null Values – 3 Valued Logic

(null > 0) is null

(null + 1) is null

(null = 0) is null

null AND true is null

AND	T	F	Null
T	T	F	Null
F	F	F	F
Null	Null	F	Null

OR	T	F	Null
T	T	T	T
F	T	F	Null
Null	T	Null	Null

SELECT (distinct)

- SQL no elimina automáticamente las tuplas duplicadas. Para hacerlo se usa DISTINCT

```
SELECT  DISTINCT dcod
FROM empleados
```

Funciones agregadas-Group by)

- Funciones: COUNT, SUM, MAX, MIN, AVG
- Operan sobre un grupo de filas
- Los grupos de filas se definen con la clausula GROUP BY
- Si el select no tiene un GROUP BY el grupo está formado por todas las filas de la tabla

Funciones agregadas-Group by)

```
db2 => select min(esalario), max(esalario)
         from empleados
1            2
-------      --------
 1000.00    2200.00
```

```
db2 => select dcod, min(esalario), max(esalario)
       from empleados group by dcod
DCOD       2          3
---------- --------   --------
        10  1000.00  1200.00
        15  1000.00  2000.00
        20  2200.00  2200.00
```

```
db2 => select dcod,avg(esalario) SAL_PRM
         from empleados group by dcod
DCOD       SAL_PRM
---------- ----------------------------------
        10  1100.00
        15  1500.00
        20  2200.00
```

```
db2 => select * from empleados
ECOD NOMBRE  DCOD   ESALARIO
-------- ------------- ---------- --------
       1 Juan          10       1000.00
       2 Pedro         15       2000.00
       3 Maria         10       1200.00
       4 Juana         20       2200.00
       5 Cata          15       1000.00
 5 registro(s) seleccionados.
```

SELECT (group by)

```
SELECT dcod, enombre, AVG(esalario)
FROM empleados
GROUP BY dcod;
```

- Es posible ?, Que devolvera ?

SELECT (group by – having)

```
SELECT dcod, count(*) , AVG(esalario)
FROM empleados
GROUP BY dcod;
```

```
SELECT dcod, count(*) , AVG(esalario)
FROM empleados
GROUP BY dcod
HAVING count(*) > 10  -- cond/restric sobre el grupo
```

SELECT (order by)

- Para ordenar las filas que retorna la consulta.
- El valor por default es ASC

```
SELECT ddescr, enombre, esalario
FROM empleados e, departamentos d
WHERE e.dcod = d.dcod
ORDER BY esalario DESC, d.dcod ASC
```

SELECT (like)

```
SELECT *
FROM empleados
WHERE enombre LIKE '%H%';
```

- Otras opciones:

```
WHERE enombre LIKE '__H_';
WHERE enombre LIKE '__H%';
```

SELECT

SELECT /* *columnas/expresiones a ser retornadas* */
FROM /* *relaciones entre tablas* */
[WHERE /* *condic sobre la filas a ser retornadas* */]
[GROUP BY /* *atributos de agrupamiento* */]
[HAVING /**cond sobre los grupos* */]
[ORDER BY /**orden en que se retornan las filas** */]

DML - UPDATE

- Modifica filas existentes en una tabla

```
UPDATE table
SET column = value [, column = value, ...]
[WHERE condition];
```

- Ejemplos

```
UPDATE empleados
 SET dcod = 20
 WHERE ecod = 7782;
```

DML - DELETE

- Borra filas existentes en una tabla

```
DELETE [FROM] table
[WHERE condition];
```

- Ejemplos

```
DELETE FROM departamentos
WHERE ddescr = 'FINANZAS';
```

Delete sin where borra todas las filas, pero la tabla permanece creada (sin filas)

Select Anidados

La clausula WHERE puede contener un Select anidado !

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid IN (SELECT R.sid
                 FROM  Reserves R
                 WHERE  R.bid=103)
```

Reserves

sid	bid	day
1	101	9/12
2	103	9/13
1	105	9/13

sname
Bilbo

Sailors

	sid	sname	rating	age
S	1	Frodo	7	22
S	2	Bilbo	2	39
S	3	Sam	8	27

Primero obtiene el conjunto de los marinos que alquilaron el bote #103…(Inner query)

…y luego para cada fila del outer query verifica si cumple la clausula IN

sid
2

Buscar los nombres de los Marinos que alquilaron el bote #103:

Consultas anidadas

```
SELECT     select_list
FROM       table
WHERE      expr operator
           (SELECT    select_list
           FROM        table);
```

- Usar single-rows operadores para subqueries que retornan una fila (=, >, <, <>, >=, <=)
- Usar multiple-rows operadores para subqueries que retornan varias filas (IN, ANY, ALL)

Consultas anidadas

```
SELECT enombre, esalario
FROM  empleados
WHERE  esalario = (SELECT MIN(esalario)
                   FROM empleados);

SELECT dcod, MIN(esalario)
FROM   empleados
GROUP BY dcod
HAVING MIN(esalario) > (SELECT AVG(esalario)
                        FROM  empleados);
```

Es responsabilidad de quien escribe el query asegurar que el subquery devolverá una sola fila. Si el subquery devuelve 0 o + de 1 fila, da error

Consultas anidadas

OPERADOR	SIGNIFICADO
IN	Retorna TRUE si está incluido en los valores retornados por el subquery
ANY	Retorna TRUE si la comparación es TRUE para al menos un valor retornado por el subquery
ALL	Retorna TRUE si la comparación es TRUE para todos los valores retornados por el subquery
EXISTS	Retorna TRUE si el subquery devuelve al menos una fila. FALSE si devuelve 0 filas

Consultas anidadas

```
SELECT  enombre, esalario
FROM    empleados
WHERE   esalario < ANY
              (SELECT esalario
                FROM  empleados
                WHERE dcod = 20);


SELECT  enombre, esalario
FROM    empleados
WHERE   esalario > ALL
              (SELECT esalario
                FROM empleados
                WHERE dcod = 20);
```

Consultas anidadas

```
SELECT  enombre, esalario
FROM    empleados
WHERE   dcod IN
             (SELECT dcod
              FROM  departamentos
              WHERE ddescr LIKE '%FINAN%');

SELECT dcod, ddescr
FROM departamentos d
WHERE NOT EXISTS (SELECT *
                  FROM empleados e
                  WHERE  d.dcod = e.dcod);
```

Consultas anidadas

```
UPDATE  empleados
       SET  (cargo, dcod) = (SELECT cargo, dcod
                             FROM empleados
                             WHERE ecod = 7499)
        WHERE ecod = 7698;

DELETE FROM empleados
WHERE  dcod =
              (SELECT   dcod
               FROM     departamentos
               WHERE    ddescr ='VENTAS');
```

Consultas anidadas

```
SELECT esalario
FROM (SELECT esalario, egeren, dcod
      FROM empleados
      WHERE egeren IS NOT NULL)
WHERE dcod = 7698;
```

SELECT (UNION)

- El operador UNION retorna las filas pertenecientes a ambas consultas eliminando las duplicadas

```
SELECT enombre, ecargo
FROM  empleados
UNION
SELECT enombre, efuncion
FROM  emp_hist;
```

SELECT (UNION ALL)

- El operador UNION retorna las filas pertenecientes a ambas consultas incluídas las duplicadas

```
SELECT enombre, ecargo
FROM  empleados
UNION ALL
SELECT enombre, efuncion
FROM  emp_hist;
```

SELECT (INTERSECT)

- El operador INTERSECT retorna las filas comunes a ambas consultas

```
SELECT enombre, ecargo
FROM  empleados
INTERSECT
SELECT enombre, efuncion
FROM  emp_hist;
```

SELECT (MINUS)

- El operador MINUS retorna las filas de la primera consulta que no están presentes en la segunda

```
SELECT enombre, ecargo
FROM  empleados
MINUS
SELECT enombre, efuncion
FROM  emp_hist;
```

Mas consultas anidadas

- Empleados que ganan más que el promedio de salarios de su departamento

```
SELECT enombre, esalario, dcod
FROM  empleados e1
WHERE esalario > (SELECT AVG(esalario)
                  FROM   empleados e2
                  WHERE  e1.dcod = e2.dcod);
```

Es un subquery Correlacionado, ya que en el subquery, se hace referencia a la tabla del query externo. Por cada fila candidata del query externo, se ejecuta el subquery para verificar si la fila pertenece al resultado.

Mas consultas anidadas

- Empleados que tienen algun empleado a cargo

```
SELECT enombre
FROM  empleados e1
WHERE EXISTS (SELECT *
              FROM   empleados e2
              WHERE  e1.ecod = e2.egeren);
```

Mas consultas anidadas

- El menor salario por departamento de aquellos con más de 7 empleados.

```
SELECT dcod, MIN(esalario)
FROM  empleados e1
GROUP BY dcod
HAVING COUNT(*) > 7
```

Mas consultas anidadas

- Actualizar el salario de los empleados de los departamentos 1020 y 1040, sumandole el ultimo premio asignado

```
UPDATE empleados e
SET esalario = (SELECT empleados.esalario + p1.premio
                FROM   premios p1
                WHERE  p1.ecod = e.ecod AND
                       p1.fecha_premio =
                        (SELECT MAX(p2.fecha_premio)
                         FROM   premios  p2
                         WHERE e.ecod=p2.ecod)
)
WHERE dcod IN ('1020', '1040');
```

VISTAS

- Son relaciones pero de las cuales solo almacenamos su definición, no su conjunto de filas.

```
CREATE VIEW view [(column [, column...])]
       AS SELECT ……;

DROP VIEW view;

CREATE VIEW depto_totales (ecod, totsal, maxsal)
   AS SELECT ecod, sum(esalario), max(esalario)
       FROM empleados
       GROUP BY ecod;
```

INDICES

- Es una estructura de acceso físico a datos
- Son usados para acceder más rapidamente a filas de tablas.
- Son independientes lógica y físicamente de la tabla que indexan

```
CREATE INDEX index
ON table (column[, column]...);

DROP INDEX index;

CREATE INDEX emp_enombre_i
ON  empleados (enombre);
```

SEGURIDAD

- GRANT: otorga privilegios sobre objetos de la DB a usuarios o roles

```
GRANT privileges ON object TO usuario/rol [WITH GRANT
                                           OPTION]
```

- Los privilegios son SELECT/INSERT/UPDATE/EXECUTE
- REVOKE: elimina privilegios sobre objetos de la DB a usuarios o roles

```
REVOKE [GRANT OPTION FOR] privileges ON object FROM
usuario/rol
```

SEGURIDAD

```
GRANT select, update ON empleados TO scott

REVOKE update ON empleados FROM scott
```

- El usuario que creo el objeto tiene todos los permisos sobre él por defecto

SQL embebido

- SQL incluído en otros lenguajes de programación (host language)
- Un precompilador convierte la sentencia SQL en una llamada a un API especial. Luego el compilador habitual compila el código.
- Variables del lenguaje host pueden ser usadas en el SQL embebido. Se las reconoce por el prefijo ":"
- Existen variables especiales que retornan el resultado del sql embebido

SQL embebido

```
EXEC SQL SELECT count(*) INTO :v_cant
         FROM empleados
         WHERE dcod = :v_depto
```

SQL embebido

EXEC SQL CONNECT /*se conecta con la base*

EXEC SQL BEGIN (END) DECLARE SECTION /*
declaracion de variables*/

EXEC SQL *statement* /*ejecuta sentencia SQL*/

SQLCODE

- Variable implícita que retorna el resultado de un comando sql embebido

  exitoso => sqlcode = 0

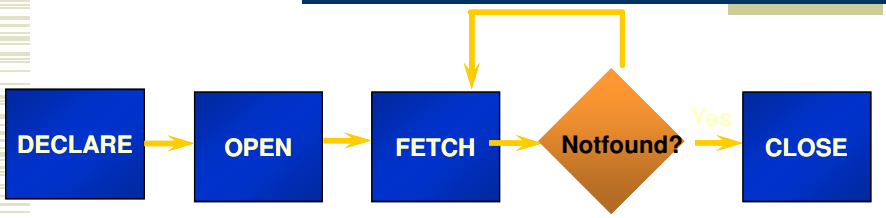
  erroneo => sqlcode < 0

  warning => sqlcode > 0

SQL embebido

- Para trabajar con conjunto de filas se usan los CURSORES
- Un cursor es un puntero que apunta a una fila que pertenece a un conjunto de registros

SQL embebido - Cursores



DECLARE CURSOR IS – define el conj de filas mediante una consulta

OPEN – resuelve la consulta y posiciona el cursor en la primera fila resultante

FETCH – avanza el cursor a la proxima fila del conj de filas resultante

CLOSE – libera el resultado de la consulta

NOT FOUND – código retornado (sqlcode) cuando el cursor pasó la ultima fila de la consulta

SQL embebido - Cursores

DECLARE CURSOR *cursor* IS SELECT … [FOR UPDATE OF *col*];

OPEN CURSOR *cursor*;

FETCH [orientacion] [FROM] *cursor* INTO *vlist*;

orientacion: next, prior, first, last, relative i, absolute i.

CLOSE *cursor*;

SQL embebido - Cursores

```
EXEC SQL DECLARE CURSOR emple_c IS SELECT enombre, esal
                                   FROM empleados;

EXEC SQL OPEN CURSOR emple_c;

EXEC SQL FETCH emple_c INTO :v_nombre, :v_sal;
WHILE SQLCODE == 0 DO
  BEGIN
      writeln(v_nombre);
      salario := salario + v_sal;
      EXEC SQL FETCH emple_c INTO :v_nombre, :v_sal;
   END;
EXEC SQL CLOSE emple_c;
```

SQL embebido - Cursores

```
EXEC SQL UDPATE  empleados
            SET sal = :v_salconaum
            WHERE ecod = :v_ecod;
```

Stored Procedures

- Es una porción de código, que se puede invocar mediante una sentencia SQL.
- Se ejecuta en el servidor de base de datos
- Encapsulan reglas de negocio fuertemente relacionadas con los datos de la BD y sin interacción con el usuario
- Permite reutilizar código
- No es obligatorio que esten escritos en SQL (Java, PL/SQL, Transact SQL)
- Cada RDBMS tiene su propio lenguaje de Stored Procedure, los cuales incluyen sentencias de control, manejo de variables, etc.
- Los stored Procedures, tienen un nombre, reciben parametros y pueden devolver resultados

Stored Procedures (Ejemplo)

```
CREATE PROCEDURE drp_depto(IN cod_depto INTEGER)
    LANGUAGE SQL
    BEGIN
        -- Antes de borrar un depto debemos mover los empleados en dicho depto a un departamento temporal
        DECLARE cod_depto_temporal integer;

        -- obtenemos el DID del depto temporal
        SELECT did INTO cod_depto_temporal FROM DEPTOS WHERE NOMBRE = 'TEMPORAL';

        -- movemos los empleados al depto temporal
        UPDATE EMPLEADOS
        SET DCOD = cod_depto_temporal
        WHERE DCOD = cod_depto;

        -- finalmente, borramos el departamento
        DELETE FROM DEPTOS WHERE DID = cod_depto;
    END
```

Se ejecuta con : CALL drp_depto(10)

Trigger

- Código almacenado en la DB que se ejecuta ante ciertos eventos.
  - Evento: activa el trigger
  - Acción: código que se ejecuta si se dispara el trigger

Trigger (Ejemplo)

```
-- create a table to use for with the trigger in this example if it has not already been created
-- previously if the table does not exist, the trigger will be invalid
CREATE TABLE emp_audit  ( emp_audit_id NUMBER(6), up_date DATE,
                          new_sal NUMBER(8,2), old_sal NUMBER(8,2) );
-- create or replace the trigger
CREATE OR REPLACE TRIGGER audit_sal
AFTER UPDATE OF salary ON employees
FOR EACH ROW
BEGIN                    -- bind variables are used here for values
INSERT INTO emp_audit VALUES( :OLD.employee_id, SYSDATE,
    :NEW.salary, :OLD.salary );
END;
-- fire the trigger with an update of salary
UPDATE employees SET salary = salary * 1.01 WHERE manager_id = 122;
-- check the audit table to see if trigger was fired
SELECT * FROM emp_audit;
```

División en SQL (con Not Exists)

Obtener los marinos que alquilaron TODOS los botes

```
SELECT  S.sname
FROM  Sailors S
WHERE NOT EXISTS
  (SELECT  B.bid
   FROM  Boats B
   WHERE  NOT EXISTS
          (SELECT  R.bid
           FROM  Reserves R
           WHERE  R.bid=B.bid
              AND R.sid=S.sid))
```

Obtener los marinos S tales que...

No existe ningún bote B …

Sin una reserva a nombre del marino S

Division

```
SELECT  S.sname
FROM  Sailors S
WHERE NOT EXISTS
  (SELECT  B.bid
   FROM  Boats B
   WHERE  NOT EXISTS
          (SELECT  R.bid
           FROM  Reserves R
           WHERE  R.bid=  103
              AND R.sid=  3   )
```

Sailors

	sid	sname	rating	age
S	1	Frodo	7	22
S	2	Bilbo	2	39
S	3	Sam	8	27

Boats

	bid	bname	color
B	101	Nina	red
B	103	Pinta	blue

Reserves

	sid	bid	day
R	1	103	9/12
R	2	103	9/13
R	3	103	9/14
R	3	101	9/12
R	1	103	9/13