# *Instrucciones SQL para utilizar con Advantage*

**Instrucciones soportadas:**

## ALTER TABLE

**Modifica la estructura de una tabla y agrega o elimina columnas.**

**Sintáxis:**

```
ALTER TABLE <tabla><acción>[<acción>…]
```

**Acciones:**

```
ADD [COLUMN] <nombre-columna> <tipo-dato(tamaño[,decimales])>
       [CONSTRAINT NOT NULL] |
       [CONSTRAINT MINIMUM <minimo-valor-columna>] |
       [CONSTRAINT MAXIMUM <maximo-valor-columna>] |
       [CONSTRAINT ERROR MESSAGE <mesaje-error>] |
       [DEFAULT <valor-default-columna>]

ALTER [COLUMN] <nombre-original-columna>  <tipo-dato(tamaño[,decimales])>
       [CONSTRAINT NOT NULL] |
       [CONSTRAINT MINIMUM <minimo-valor-columna>] |
       [CONSTRAINT MAXIMUM <maximo-valor-columna>] |
       [CONSTRAINT ERROR MESSAGE <mesaje-error>] |
       [DEFAULT <valor-default-columna>]

ALTER [COLUMN] <nombre-columna>
       DROP NOT NULL | MINIMUM | MAXIMUM | ERROR MESSAGE | DEFAULT
       [DROP [COLUMN] <nombre-columna> ]
       [DROP [CONSTRAINT] PRIMARY KEY ]
```

**Observaciones:**

**Si la tabla esta en una base de datos, es decir, si está asociada a una Diccionario de Datos Advantage, la tabla unicamente puede ser alterada desde el diccionario desde la cuenta de administrador (ADSSYS).**

**Si se elimina una llave primaria (primary key), se le quita al índice el atributo de llave primaria, pero el índice permanece sin cambios, si desea borrar el índice utilice la instrucción DROP INDEX.**

**Ejemplo:**

```
ALTER TABLE salesreps ADD COLUMN region char(40)
```

```
ALTER TABLE customers ADD address3 char(40) ADD COLUMN refund integer

ALTER TABLE orders ALTER COLUMN price price curdouble

ALTER TABLE offices ALTER COLUMN mgr mgr integer DEFAULT '104'

ALTER TABLE salesreps ALTER name name char(40) CONSTRAINT NOT NULL

ALTER TABLE demo10
       ADD COLUMN test char(20)
                  DEFAULT ‘abcde’
                  CONSTRAINT MINIMUM ‘A’
                  CONSTRAINT MAXIMUM ‘z’
                 CONSTRAINT NOT NULL
                 CONSTRAINT error message
                        ‘A bad value was input for the column named test’
       ADD COLUMN test2 SHORT
               DEFAULT ‘45’
               CONSTRAINT MINIMUM ‘2’
               CONSTRAINT MAXIMUM ‘169’
               CONSTRAINT NOT NULL
               CONSTRAINT error message
                   ‘An invalid values was set in the column named test2’
       ALTER COLUMN lastname lastnamechanged char(40)
         DEFAULT ‘Donahue’
               CONSTRAINT MINIMUM ‘A’
               CONSTRAINT MAXIMUM ‘Z’
               CONSTRAINT error message
                        ‘An invalid lastnamechanged was input’
       ALTER COLUMN firstname firstnamechanged char(30)
       DROP deptnum
       DROP dateofhire

ALTER table demo10
        ALTER COLUMN lastnamechanged DROP DEFAULT
        ALTER COLUMN lastnamechanged DROP MINIMUM
        ALTER COLUMN lastnamechanged DROP MAXIMUM


        ALTER COLUMN lastnamechanged DROP error message
        ALTER COLUMN lastnamechanged DROP NOT NULL


ALTER TABLE table1 DROP CONSTRAINT PRIMAY KEY


ALTER TABLE table1 DROP PRIMARY KEY
```

## CREATE INDEX

**Crea un índice para una tabla.**

**Sintáxis**

```
CREATE [UNIQUE] INDEX <nombre-indice> ON <tabla>
       ( <nombre-columna>[ASC | DESC],... )
```

**Observaciones:**

**ASC (ascendente) es el valor por default, DESC (descendente) debe de ser textualmente indicado si se desea un índice de mayor a menor.**

**Si la tabla es una tabla de base de datos, es decir, si la table esta asociada con un Diccionario de Datos Advantage, la instrucción CREATE INDEX solo puede ser ejecutada desde el administrador del diccionario (ADSSYS). El nuevo índice creado se abrirá automaticamente y estará disponible para todos los usuarios de la base de datos.**

**Ejemplo:**

```
CREATE INDEX empndx ON emp ( emp_addr, emp_name )

CREATE UNIQUE INDEX empuniq ON emp ( emp_id  DESC)
```

## CREATE PROCEDURE

**Añade un Procedimiento Extendido Advantage (procedimiento almacenado) en el Diccionario de Datos Advantage.**

**Sintáxis:**

```
CREATE PROCEDURE <nombre-procedimiento>
       ( [<nombre-parametro> <tipo-dato> [OUTPUT]]
         [,<nombre parametro> <tipo-dato> [OUTPUT]] …)
FUNCTION <nombre-dll-funciones> IN LIBRARY < biblioteca-AEP >
```

**Observaciones:**

**<nombre-dll-funciones> es el nombre de la función en el archivos de Procedimientos Extendidos Advantage que debe ser invocado cuando el procedimiento llamado <nombre-procedimiento> es ejecutado.**

**<biblioteca-AEP> es el nombre del archioe de Procedimientos Extendidos Advantage. Este archivo es una DLL de Win32. El valor de la extension por default es AEP. Si el archivo AEP tiene diferente extension o si el archvio existe en un directorio diferente que el del Diccionario de Datos Advantage, se deben realizar 2 accoines: Primero, el nombre completo y la ruta deben estar encerrados entre comillas dobles (“) o entre corchetes []. Segundo, la ruta no debe contener letras de unidades disco. La instrucción SQL y el procedimiento almacenado son ejecutado desde el Servidor Advantage por lo tanto se deberá utilizar la UNC para ubicar el archivo. Un ejemplo relativo a esto sería:**

**Asumamos que el diccionario de dato esta localizado en :**

**\\server1\datashare\database\sample.add**

**Asumamos que el procedimiento almacenado llamado SP.AEP esta localizado en:**

**\\server1\datashare\database\stored_procs\sp.aep**

**La ruta relative sería:**

**stored_procs\sp.aep**
**ó**
**\stored_procs\sp.aep**

**La palabra OUTPUT indica que el parámetro es un parámetro de salida. Los otros parámetros son parámetros de entrada. Los parámetros de entrada deben ser proporcionados en la llamada al comando EXECUTE PROCEDURE y serán pasados al procedimiento almacenado. Si el procedimiento almacenado tiene parametros de salida, la llamada a EXECUTE PROCEDURE devolverá un cursor que contiene los parametros de salida en columnas. El cursor puede consistir de múltiples renglones.**

**La instrucción CREATE PROCEDURE solo puede ser ejecutada desde la cuenta de administrador del diccionario de datos (ADSSYS).**

**Ejemplo:**

```
CREATE PROCEDURE BeginTransaction()
FUNCTION BeginTransaction IN LIBRARY "Ejemplo4StoredProc.aep"

CREATE PROCEDURE CommitTransaction()
FUNCTION CommitTransaction IN LIBRARY "Ejemplo4StoredProc.aep"

CREATE PROCEDURE RollBackTransaction()
FUNCTION RollBackTransaction IN LIBRARY "Ejemplo4StoredProc.aep"

CREATE PROCEDURE GetInfoForTable
(  RecordCount INTEGER OUTPUT,
   MaxEmployeeID INTEGER OUTPUT,
   NewestEmployee CHAR(41) OUTPUT,
   UniqueLastNameCount INTEGER OUTPUT )
FUNCTION GetInfoForTable IN LIBRARY "Ejemplo4StoredProc.aep"

CREATE PROCEDURE AddRecordToData
( LastName  CHAR(20),
   FirstName CHAR(20),
   EmpID     SHORT,
   Married   LOGICAL )
FUNCTION AddRecordToData IN LIBRARY "Ejemplo4StoredProc.aep"

CREATE PROCEDURE MyComAEP
( Lastname CHAR(20) )
```

```
FUNCTION MyProcedure IN LIBRARY "ClassLibrary1.ComClass1"
```

## CREATE TABLE

**Crea una nueva tabla en la base de datos.**

**Sintáxis:**

```
CREATE TABLE <tabla>
      ( <nombre-columna> <tipo-dato (tamaño[,tamaño])>
            CONSTRAINT NOT NULL |
            CONSTRAINT MINIMUM <maximo-valor-columna> |
            CONSTRAINT MAXIMUM <minimo-valor-columna> |
            CONSTRAINT ERROR MESSAGE <mesaje-error> |
            DEFAULT <valor-default-columna> |
            [CONSTRAINT <nombre-indice>] PRIMARY KEY]
      [,<nombre-columna> <tipo-dato (tamaño[,tamaño])>
            CONSTRAINT NOT NULL |
            CONSTRAINT MINIMUM <maximo-valor-columna> |
            CONSTRAINT MAXIMUM <minimo-valor-columna> |
            CONSTRAINT ERROR MESSAGE <mensaje-error> |
            DEFAULT <valor-default-columna> |
            [CONSTRAINT <nombre-indice] PRIMARY KEY]...] )
      [CONSTRAINT <nombre-indice>]
            PRIMARY KEY ( <nombre-columna> [DESC|ASC]
                        [,<nombre-columna> [DESC|ASC]…] )
```

**Observaciones:**

**Columnas únicas (UNIQUE) se construyen a partir de indices únicos.**

**Si la instrucción CREATE TABLE es ejecutada por el administrador del diccionario (ADDSYS) en una conexión a una base de datos, la tabla creada será parte del la base de datos definida en el diccionario. Si la instruccio CREATE TABLE es ejecutada desde una cuenta de usuario común y corriente en una conexión a la base datos, la tabla nueva será un archivo independiente.**

**Las restricciones a nivel columna solo pueden ser especificadas si la instrucción es ejecutada desde el administrador del diccionario (ADSSYS) en una conexión a la base de datos. Se generará un error si se especifican restricciones y la instrucción no es ejecutada desde la cuenta de administrador.**

**La definición de llaves primarias solo es soportada por tables ADt puede puede ser realizada de dos maneras: como una restricción definida por columna o bien por una restricción de la tabla que es definida como un elemento separado de la lista de defincciones de columnas. Cuando se define una llave primaria, un índice único es construido utilizado el campo o los campos desigando y luego es añadido al archivo índice. Si una conexión libre se utilizó para crear la tabla, esta será un archivo independiente y se creará un archivo índice pero no se creará una llave primaria ya que esta funcionalidad se limita al uso de diccionarios de datos. Si se utiliza la restricción (CONSTRAINT <nombre-índice>) en frente de la palabra PRIMARY KEY, el índice que se crea se llamará <nombre-índice>. Si esta sintáxis no es utilizada,**

**Advantage lo llamará "pk_<tabla>" donde <tabla> is the nombre de la tabla recién creada.**

**Ejemplo:**

```
CREATE TABLE sal (emp_id short, salary double, dept char(20))

CREATE TABLE emp (emp_id short,
                  name char(50),
                  hire_date date,
                  married logical)

CREATE TABLE test (
        field1 INTEGER
                DEFAULT '45'
                CONSTRAINT MINIMUM '2'
                CONSTRAINT MAXIMUM '169'
                CONSTRAINT NOT NULL
                CONSTRAINT ERROR MESSAGE 'you entered a bad value.',
        field2 CHAR(2) )

CREATE TABLE table1 ( field1 CHAR(10),
                      field2 INTEGER PRIMARY KEY )

CREATE TABLE table1 ( field1 CHAR(10),
                      field2 INTEGER,
                      PRIMARY KEY( field1, field2 DESC ) )

CREATE TABLE table1 ( field1 INTEGER CONSTRAINT index1 PRIMARY KEY )

CREATE TABLE table1 ( filed1 INTEGER,
                      CONSTRAINT index1 PRIMARY KEY( field1 ASC ) )
```

## CREATE VIEW

**Crea una nueva vista de la base de datos.**

**Sintáxis:**

```
CREATE VIEW <nombre-vista> AS SELECT <instruccion-select>
```

**Observaciones:**

**La instrucción SELECT no puede utilizar la clausula ORDER BY. Las vistas son por definición tablas virtuales que no están ordenadas.**

**Las vistas está soportadas através del uso del Diccionario de Datos Advantage. Las instrucciones CREATE VIEW deben ser ejecutadas en una conexión valida del diccionario de datos y desde la cuenta de administrador.**

**Ejemplo:**

```
CREATE VIEW GoodCustomers
       AS SELECT cust_name
            from customers
            where credit_limit > 50000
```

## DELETE

**Elimina un registro de una tabla en una base de datos.**

**Sintáxis:**

```
DELETE FROM <tabla> [WHERE <condicion-búsqueda>]
```

**Observaciones:**

**No se soportan borrados por posicionamiento.**

**El formato {ts/d/t} no se require en la condición de búsqueda para los tipos de datos timestamp, date ó time.**

**Si la instrucción DELETE es ejecutada desde una conexión a la base de datos y dicha base de datos ha sido configurada para verificación de derechos para los usuarios, para poder ejecutar el query de manera exitosa, el usuario debe tener derechos de escritura (WRITE) a la tabla.**

**Cuando se borran múltiples registros, el motor de SQL bloquea los registros mientras los borra. Intenta varias veces bloquear el registro, pero si falla en el bloqueo, continúa e intenta completar el resto de las operaciones de borrado. Si esto sucede el error (5003) se retorna al cliente, para asegurar el correcto borrado de todos los registros, se sugiere colocar la instrucción DELETE dentro de una transacción.**

**Ejemplo:**

```
DELETE FROM customer WHERE purch_amt < 100.00 AND NOT state = 'CA'
DELETE FROM customer WHERE purch_date < {d '1980-07-01'}
DELETE FROM customer WHERE purch_date < '1980-07-01'
DELETE FROM customer WHERE customer.id NOT IN
(SELECT custid FROM orders WHERE orders.custid = customer.id )
```

## DROP INDEX

**Borra los ordenes del índice (tags) en un indice CDX o ADI. Cuando el último orden del índice es borrado, el archivo índice es removido.**

**Sintáxis:**

```
DROP INDEX <tabla>.<nombre-indice>
```

**Observaciones:**

**Si la table es una tabla de una base de datos, esto es, si la tabla está asociada con un diccionario de datos, la instrucción DROP INDEX solo puede ser ejecutad desde la cuenta de administrador del Diccionario (ADSSYS).**

**Ejemplo:**

```
DROP INDEX emp.empid
```

## DROP PROCEDURE

**Elimina la definición de un Procedimiento Extendido Advantage (procedimiento almacenado) del diccionario de datos Advantage.**

**Sintáxis:**

```
DROP PROCEDURE <nombre-procedimiento>
```

**Observaciones:**

**Esta instrucción solo peude ser ejecutgada dede la cuenta de administrador del diccionario (ADSSYS).**

**Ejemplo:**

```
DROP PROCEDURE GetInfoForTable
```

## DROP TABLE

**Borra una tabla y todos sus indices asociados de la base de datos.**

**Sintáxis:**

```
DROP TABLE <tabla>
```

**Observaciones:**

**Las instrucciones CASCADE y RESTRICT no son soportadas.**

**Si la tabla es una tabla perteneciente a una base de datos, es decir, si la tabla está asociada a un Diccionario de Datos Advantage, esta instrucción solo puede ser ejecutada desde la cuenta de administrador del diccionario (ADSSYS).**

**Ejemplo:**

```
DROP TABLE sal
```

## DROP VIEW

**Borra una vista de la base de datos.**

**Sintáxis:**

```
DROP VIEW <nombre-vista>
```

**Observaciones:**

**Las instrucciones CASCADE y RESTRICT no son soportadas.**

**Si la tabla es una tabla perteneciente a una base de datos, es decir, si la tabla está asociada a un Diccionario de Datos Advantage, esta instrucción solo puede ser ejecutada desde la cuenta de administrador del diccionario (ADSSYS).**

**Ejemplo:**

```
DROP VIEW GoodCustomers
```

## EXECUTE PROCEDURE

**Ejecuta un Procedimiento Extendido Advantage (procedimiento almacenado).**

**Sintáxis:**

```
EXECUTE PROCEDURE <nombre-procedimiento>
                  ( [<valor-parametro>[, <valor-parametro>]…)
```

**Ejemplo:**

```
EXECUTE PROCEDURE AddRecordToData( ‘Smith’, ‘John’, 13, TRUE )

; notese que este ejemplo require que los cuatro parametros sean
; asignados antes de que se ejecute la instrucción.
EXECUTE PROCEDURE AddRecordToData( :lname, :fname, :id, :married )

;Este ejemplo no tiene parámetros.
EXECUTE PROCEDURE BeginTransaction()
```

## GRANT

**Instrucción no soportada.**

**Observaciones:**

**La seguridad de Advantage está basada en lo siguiente:**

**La seguridad de la red para el acceso al archivo**

**Ó**

**Solo la aplicación Advantage puede abrir el archivo.**

## INSERT

**Agrega renglones de datos a la tabla**

**Sintáxis:**

```
INSERT INTO <tabla> [(<nombre-columna>[, <nombre-columna>]…)]
                  VALUES (<valor>[, <valor>]…)

ó

INSERT INTO <tabla> [(<nombre-columna>[, <nombre-columna>]…)]
                  SELECT [ALL | DISTINCT] <lista>
                        FROM <tabla>
                        [WHERE <condicion-busqueda>]
                        [GROUP BY <agrupacion>[, <agrupación]…]
                        [HAVING <condicion-busqueda>]
```

**Ejemplo:**

```
INSERT INTO sal VALUES (34086, 'Chris Isaac', 45000.00, '1992-05-25')

INSERT INTO sal (emp_id, salary) VALUES (21586, 31500.50)

INSERT INTO cust_report SELECT DISTINCT * FROM cust WHERE state = ‘CA’

INSERT INTO ts VALUES( {ts '1999-03-19 13:45:33.013'} )

INSERT INTO ts VALUES( now() )


INSERT INTO expire (expiretime) VALUES( TIMESTAMPADD( SQL_TSI_DAY, 30,
now() ))
```

## REVOKE

**No soportada**

## ROLLBACK WORK

**No soportada**

## SELECT

**Recupera datos de una base de datos.**

### Sintaxis:

```
SELECT [{static}] [ALL | DISTINCT] <lista-campos>
        [INTO nombretabla]
        FROM   nombre-tabla | nombre-tabla nombre-alias
               INNER JOIN nombre-tabla | nombre-tabla nombre-alias ON booleano | inner-join
               INNER JOIN nombre-tabla | nombre-tabla nombre-alias
                      ON Booleano | nombre-tabla | nombre-tabla nombre-alias
               LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                      ON booleano | nombre-tabla | nombre-tabla nombre-alias
               LEFT [OUTER] JOIN outer-join ON booleano | outer-join
               LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias ON booleano |
               { OJ nombre-tabla | nombre-tabla nombre-alias
                      LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                             ON booleano | nombre-tabla | nombre-tabla nombre-alias
                      LEFT [OUTER] JOIN outer-join ON booleano | outer-join
                      LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                             ON booleano}
               ,
               lista-tablas | nombre-tabla | nombre-tabla nombre-alias
               INNER JOIN nombre-tabla | nombre-tabla nombre-alias ON booleano | inner-join
               INNER JOIN nombre-tabla | nombre-tabla nombre-alias
                      ON Booleano | nombre-tabla | nombre-tabla nombre-alias
               LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                      ON booleano | nombre-tabla | nombre-tabla nombre-alias
               LEFT [OUTER] JOIN outer-join ON booleano | outer-join
               LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias ON booleano |
               { OJ nombre-tabla | nombre-tabla nombre-alias
                      LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                             ON booleano | nombre-tabla | nombre-tabla nombre-alias
                      LEFT [OUTER] JOIN outer-join ON booleano | outer-join
                      LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                             ON booleano}
        [WHERE <condicion-busqueda>]
        [GROUP BY <columna-agrupadora> [, < columna-agrupadora]…]
        [HAVING <condicion-busqueda>]
        [ORDER BY {<nombre-columna> | integer} [ASC | DESC]
               [,{<nombre-columna> | integer} [ ASC | DESC ]]…]

ó

SELECT [{static}] [ALL | DISTINCT] <lista-campos>
        [INTO nombretabla]
        FROM   nombre-tabla | nombre-tabla nombre-alias
```

```
            INNER JOIN nombre-tabla | nombre-tabla nombre-alias ON booleano | inner-join
            INNER JOIN nombre-tabla | nombre-tabla nombre-alias
                   ON Booleano | nombre-tabla | nombre-tabla nombre-alias
            LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                   ON booleano | nombre-tabla | nombre-tabla nombre-alias
            LEFT [OUTER] JOIN outer-join ON booleano | outer-join
            LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias ON booleano |
            { OJ nombre-tabla | nombre-tabla nombre-alias
                   LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                          ON booleano | nombre-tabla | nombre-tabla nombre-alias
                   LEFT [OUTER] JOIN outer-join ON booleano | outer-join
                   LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                          ON booleano}
      ,
            lista-tablas | nombre-tabla | nombre-tabla nombre-alias
            INNER JOIN nombre-tabla | nombre-tabla nombre-alias ON booleano | inner-join
            INNER JOIN nombre-tabla | nombre-tabla nombre-alias
                   ON Booleano | nombre-tabla | nombre-tabla nombre-alias
            LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                   ON booleano | nombre-tabla | nombre-tabla nombre-alias
            LEFT [OUTER] JOIN outer-join ON booleano | outer-join
            LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias ON booleano |
            { OJ nombre-tabla | nombre-tabla nombre-alias
                   LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                          ON booleano | nombre-tabla | nombre-tabla nombre-alias
                   LEFT [OUTER] JOIN outer-join ON booleano | outer-join
                   LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                          ON booleano}
      [WHERE <condicion-busqueda>]
      [GROUP BY <columna-agrupadora> [, < columna-agrupadora]…]
      [HAVING <condicion-busqueda>]
      UNION [ALL] [{static}] [ALL | DISTINCT] <lista-campos>
            [INTO nombretabla]
            FROM   nombre-tabla | nombre-tabla nombre-alias
                   INNER JOIN nombre-tabla | nombre-tabla nombre-alias ON booleano |
inner-join
                   INNER JOIN nombre-tabla | nombre-tabla nombre-alias
                          ON Booleano | nombre-tabla | nombre-tabla nombre-alias
                   LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                          ON booleano | nombre-tabla | nombre-tabla nombre-alias
                   LEFT [OUTER] JOIN outer-join ON booleano | outer-join
                   LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias ON
booleano |
                   { OJ nombre-tabla | nombre-tabla nombre-alias
                          LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                                 ON booleano | nombre-tabla | nombre-tabla nombre-alias
                          LEFT [OUTER] JOIN outer-join ON booleano | outer-join
                          LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                          ON booleano}
                   ,
                   lista-tablas | nombre-tabla | nombre-tabla nombre-alias
                   INNER JOIN nombre-tabla | nombre-tabla nombre-alias ON booleano |
inner-join
                   INNER JOIN nombre-tabla | nombre-tabla nombre-alias
```

```
                        ON Booleano | nombre-tabla | nombre-tabla nombre-alias
                    LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                        ON booleano | nombre-tabla | nombre-tabla nombre-alias
                    LEFT [OUTER] JOIN outer-join ON booleano | outer-join
                    LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias ON
booleano |
                    { OJ nombre-tabla | nombre-tabla nombre-alias
                        LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                            ON booleano | nombre-tabla | nombre-tabla nombre-alias
                        LEFT [OUTER] JOIN outer-join ON booleano | outer-join
                        LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                        ON booleano}
          [WHERE <condicion-busqueda>]
          [GROUP BY <columna-agrupadora> [, < columna-agrupadora]…]
          [HAVING <condicion-busqueda>]
        [ UNION [ALL] [{static}] [ALL | DISTINCT] <lista-campos>
              [INTO nombretabla]
              FROM    nombre-tabla | nombre-tabla nombre-alias
                    INNER JOIN nombre-tabla | nombre-tabla nombre-alias ON booleano |
inner-join
                    INNER JOIN nombre-tabla | nombre-tabla nombre-alias
                        ON Booleano | nombre-tabla | nombre-tabla nombre-alias
                    LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                        ON booleano | nombre-tabla | nombre-tabla nombre-alias
                    LEFT [OUTER] JOIN outer-join ON booleano | outer-join
                    LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias ON
booleano |
                    { OJ nombre-tabla | nombre-tabla nombre-alias
                        LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                            ON booleano | nombre-tabla | nombre-tabla nombre-alias
                        LEFT [OUTER] JOIN outer-join ON booleano | outer-join
                        LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                        ON booleano}
                    ,
                    lista-tablas | nombre-tabla | nombre-tabla nombre-alias
                    INNER JOIN nombre-tabla | nombre-tabla nombre-alias ON booleano |
inner-join
                    INNER JOIN nombre-tabla | nombre-tabla nombre-alias
                        ON Booleano | nombre-tabla | nombre-tabla nombre-alias
                    LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                        ON booleano | nombre-tabla | nombre-tabla nombre-alias
                    LEFT [OUTER] JOIN outer-join ON booleano | outer-join
                    LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias ON
booleano |
                    { OJ nombre-tabla | nombre-tabla nombre-alias
                        LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                            ON booleano | nombre-tabla | nombre-tabla nombre-alias
                        LEFT [OUTER] JOIN outer-join ON booleano | outer-join
                        LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                        ON booleano}
              [WHERE <condicion-busqueda>]
              [GROUP BY <columna-agrupadora> [, < columna-agrupadora]…]
              [HAVING <condicion-busqueda>]…]
        [ORDER BY {integer} [ASC|DESC][,{integer}[ ASC | DESC ]]…]
```

Ó

```
SELECT TOP {integer} | TOP {integer} PERCENT
        [{static}] [ALL | DISTINCT]
        <lista-campos>
        [INTO nombretabla]
        FROM   nombre-tabla | nombre-tabla nombre-alias
               INNER JOIN nombre-tabla | nombre-tabla nombre-alias ON booleano | inner-join
               INNER JOIN nombre-tabla | nombre-tabla nombre-alias
                      ON Booleano | nombre-tabla | nombre-tabla nombre-alias
               LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                      ON booleano | nombre-tabla | nombre-tabla nombre-alias
               LEFT [OUTER] JOIN outer-join ON booleano | outer-join
               LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias ON booleano |
               { OJ nombre-tabla | nombre-tabla nombre-alias
                      LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                             ON booleano | nombre-tabla | nombre-tabla nombre-alias
                      LEFT [OUTER] JOIN outer-join ON booleano | outer-join
                      LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                             ON booleano}
               ,
               lista-tablas | nombre-tabla | nombre-tabla nombre-alias
               INNER JOIN nombre-tabla | nombre-tabla nombre-alias ON booleano | inner-join
               INNER JOIN nombre-tabla | nombre-tabla nombre-alias
                      ON Booleano | nombre-tabla | nombre-tabla nombre-alias
               LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                      ON booleano | nombre-tabla | nombre-tabla nombre-alias
               LEFT [OUTER] JOIN outer-join ON booleano | outer-join
               LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias ON booleano |
               { OJ nombre-tabla | nombre-tabla nombre-alias
                      LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                             ON booleano | nombre-tabla | nombre-tabla nombre-alias
                      LEFT [OUTER] JOIN outer-join ON booleano | outer-join
                      LEFT [OUTER] JOIN nombre-tabla | nombre-tabla nombre-alias
                             ON booleano}
        [WHERE <condicion-busqueda>]
        [GROUP BY <columna-agrupadora> [, < columna-agrupadora]…]
        [HAVING <condicion-busqueda>]
        [ORDER BY {<nombre-columna> | integer} [ASC | DESC]
               [,{<nombre-columna> | integer} [ ASC | DESC ]]…]
```

**Observaciones:**

**La secuencia de escape opcional {static} puede ser usada en una instrucción SELECT para forzar el uso de un cursor estático**

**La claúsula opcional [INTO nombretabla] puede ser utilizada solamente en una vez en cada instrucción y debe aparecer en el primer SELECT de la instrucción SQL, no puede ser utilizado en un subquery. Cuando la clausula INTO es especficada, el resultado será almacenado en la tabla especificada y se creará un cursor dinámico. La nueva tabla se llenará con los datos antes de que el control sea retornado al cliente.**

**Si la instrucción SELECT es ejecutada desde una conexión a una base de datos y la base de datos ha sido configurada para verificar los drechos del usuario, para poder ejecutar el Quero, el usuario debe tener derechos de lentrua a todas las tablas y vistas que se especifique en la claúsula FROM de la instrucción**

**La claúsla TOP retorna los primeros registros de el cojunto resultante. "TOP x", en una instrucción SELECT devolverá los primeros "x" registros en el query. "TOP x PERCENT" retornará el "x" porciento de los registros. El uso de esta claúsula reduce el tiempo que el servidor toma para recuperar los datos en la mayoría de los casos. La excepción a este comportamiento es cuando se incluye la claúsula ORDER BY. Esto se debe a que el servidor debe llenar el cursor entero en orden para deteminar cual registro debe ir primero y cual después. El uso de estas claúsulas devuelven cursores estáticos.**

**Ejemplos:**

```
SELECT * FROM emp

SELECT DISTINCT lastname, firstname, idname FROM emp

SELECT * FROM emp WHERE lastname = 'Smith'

SELECT salesrep FROM sales GROUP BY emp_id HAVING sales > 10000

SELECT * FROM sal ORDER BY emp_id

SELECT * FROM emp WHERE hire_date < '1992-02-02'

SELECT * FROM emp WHERE hire_date < {d '1992-02-02'}

SELECT * FROM emp WHERE hire_date > '1993-01-01' AND
                        hire_date < {d '1993-01-01'} + 30

SELECT * FROM emp WHERE hire_date  - quit_date > 30 ORDER BY lastname

SELECT * FROM emp LEFT OUTER JOIN sal ON emp.emp_id = sal.emp_id

SELECT * FROM emp WHERE emp.emp_id IN
                     (SELECT sal.emp_id FROM sal WHERE sal.dept <> 'HR' )

SELECT * FROM oldcust WHERE oldcust.sales > 50000 UNION ALL
                        SELECT * FROM newcust WHERE newcust.sales > 50000

SELECT emp.lastname, sal.salary, dept.deptname FROM emp
       LEFT OUTER JOIN sal ON emp.emp_id = sal.emp_id, dept
       WHERE emp.dept_id=dept.dept_id

SELECT emp.lastname, sal.salary, dept.deptname
   INTO lastname_sal
   FROM emp LEFT OUTER JOIN sal ON emp.emp_id = sal.emp_id,
   WHERE emp.dept_id=dept.dept_id
```

```
SELECT TOP 100 * FROM emp

SELECT TOP 15 PERCENT lastname, firstname, idname FROM emp

SELECT TOP 10 DISTINCT lastname, firstname, idname FROM emp
```

**Funciones soportadas:**

| | |
|---|---|
| `AVG( expr )` | Calcula el promedio de un grupo de valores |
| `COUNT( expr )` | Devuelve el numero de registros en la columna que no tengan valor NULL y que satifacen la condición de búsqueda. `COUNT(*)` Devuelve el número de renglones que contiene la tabla actualmente. |
| `MAX( expr )` | Devuelve el valor máximo de un conjunto de valores. |
| `MIN( expr )` | Devuelve el valor mínimo de un conjunto de valores. |
| `SUM( expr )` | Totaliza los valores dentro de un conjunto de valores numéricos. |

**Ejemplos:**

```
SELECT AVG(quota),  AVG(sales) FROM salesinfo
SELECT AVG(100 * (sales/quota)) FROM salesinfo

SELECT COUNT(empid) FROM employees
SELECT COUNT(ordernum) FROM orders WHERE amount > 1000
SELECT COUNT(*) FROM orders WHERE amount > 1000

SELECT MIN(quota),  MAX(quota) FROM salesinfo
SELECT MAX(100 * (sales/quota)) FROM salesinfo
SELECT MIN(100 * (sales/quota)) FROM salesinfo

SELECT SUM(quota),  SUM(sales) FROM salesinfo
SELECT SUM(sales - quota) FROM salesinfo
```

## UPDATE

**Actualiza los renglones existentes en una tabla con información nueva.**

**Sintáxis:**

```
UPDATE <tabla>
       SET <nombre-columna > = {NULL | expresión | ( SELECT select )}
          [,<nombre-columna > = {NULL|expresión|( SELECT select )}]…
       [FROM lista-tablas ]
            [WHERE <condicion-busqueda>]
```

**Observaciones:**

**Si la instrucción UPDATE se ejecuta en una conexión a una base de datos y la base de datos ha sido configurada para verificar los derechos de acceso del usuario, para para poder ejecutar el query, el usuario debe tener permisos de actualización a la tabla y a las columnas que se deseen modificar.**

**Una vista tambiénn puede ser actualizada si está activa y el usuario tiene permisos de acceso a dicha vista.**

**Si existe una clausula FROM, la expresión que proporciona el nuevo valor, puede ser construida usando valores literales y columnas de las tablas especificadas en la claúsula FROM. Si no existe claúsula FROM la expresión puede ser construída usando valores literales y columnas de la tabla especificada en la instrucción UPDATE**

**La claúsula FROM especifica las tablas que proveen el criterio o los valores para la operación de actualización. Si la tabla que se está actualizando es la misma que la tabla indicada en la clausula FROM, un alias para la tabla puede especificado o no. Si la tabla que se está actualizando aparece mas de una vez en la claúsula FROM, una (y solo una) referencia a la table no debe especificar el alias de la tabla. Todas las otras referencias a la tabla en la claúsula from deben incluir una alias de tabla. Por ejemplo, lo siguiente es válido:**

```
UPDATE stock SET stock.quantity = t.quantity
       FROM stock INNER JOIN stock t ON stock.id = t.id
       WHERE t.state = 1
```

**Esto no es válido:**

```
UPDATE stock SET stock.quantity = t.quantity
       FROM stock t1 INNER JOIN stock t ON t1.id = t.id
       WHERE t.state = 1
```

**Otra forma válida de hacer la acualización sería esta:**

```
UPDATE t1 SET t1.quantity = t.quantity
       FROM stock t1 INNER JOIN stock t ON t1.id = t.id
       WHERE t.state = 1
```

**Cuando se actualizan multiples registros, el motor SQL bloquea el registro conforme realiza las actualizaciones. Realiza varios intentos de bloquear el registro. Si el bloqueo falla, continúa y trata de completar el resto de las actualizaciones en los registros pendientes. Si esto sucede, se devuelve al cliente el error 5053. Para asegurar que todos los registros sean actualizados, será necesario utilizar una transacción.**

**Ejemplo:**

```
UPDATE sal SET salary = 35000.00 WHERE emp_id = 25089

UPDATE sal SET salary = 20000, lastname = 'Jones' WHERE emp_id = 370

UPDATE sal SET salary = 20000 WHERE hire_date < '1992-02-14'

UPDATE sal SET salary = 20000 WHERE hire_date < {d '1995-05-10'}
```

```
UPDATE sal SET salary = 20000
       WHERE hire_date > '1993-01-01'
            AND hire_date < {d '1993-01-01'} + 30

UPDATE sal SET salary = 20000 WHERE hire_date - quit_date > 30

UPDATE workorder SET timedone = {ts '1999-03-19 13:45:33.013'}
```

## Funciones que se pueden utilizar en el Streamline SQL

### Funciones matemáticas:

| | |
|---|---|
| `ABS( num )` | Devuelve el valor absoluto de un numérico. |
| `ACOS( float )` | Devuelve el arcocoseno de un ángulo expresado en radianes. |
| `ASIN( float )` | Devuelve el arcoseno de un valor de un ángulo expresado en radianes. |
| `ATAN( float )` | Devuelve el arcotangente de un ángulo expresdo en radianes. |
| `ATAN2( float1, float2 )` | Devuelve el arcotangente de las coordenadas X y Y especificado por los valores float1 y float2 de un ángulo, expreado en radianes. |
| `CEILING( num )` | Devuelve el entero mas pequeño mayor o igual al valor num. |
| `COS( float )` | Devuelve el coseno de un ángulo expresado en radianes. |
| `COT( float )` | Devuelve la cotangente de un ángulo expresado en radianes. |
| `DEGREES( num )` | Devuelve el numero de grádos convertidos a partir de radianes. |
| `EXP( float )` | Devuelve el valor exponencial |
| `FLOOR( num )` | Devuelve el entero mas grande menor igual a num. |
| `LOG( float )` | Devuelve el logaritmo natural |
| `LOG10( float )` | Devuelve el logartimo base 10. |
| `MOD( int1, int2 )` | Devuelve el “modulo” (residuo) de la division de int1 entre int2. |
| `PI( )` | Devuelve el valor de PI como un valor tipo float. |
| `POWER( num, int )` | Devuelve el valor de num elevado a la potencia int |
| `RADIANS( num )` | Devuelve el numero de radianes convertidos a partir de grados. |
| `RAND( [int] )` | Devuelve un número aleatorio de punto flotante usando un valor opcional como semilla. |

| | |
|---|---|
| `ROUND( num, int )` | Devuelve el valor de num redondeado a int lugares a la derecha del punto decimal. Si int es negativo, num es redondeado a ABS(int) lugares a la izquierda del punto decimal. |
| `SIGN( num )` | Si num es menor que cero, devuelve -1. Si num es igual a 0, se devuelve 0. Si el valor es mayor a 0 se devuelve un 1. |
| `SIN( float )` | Devuelve el seno de un ángulo expresado en radianes. |
| `SQRT( float )` | Devuelve la raíz cuadrada de float. |
| `TAN( float )` | Devuelve la tangente de un ángulo expresada en radianes. |
| `TRUNCATE( num, int )` | Devuelve num truncadoa int lugares a la drecha del punto decimal. Si int es negativo, num es truncado a ABS(int) lugares al izquierda del punto decimal. |

### Funciones de manipulación de caracteres

| | |
|---|---|
| `ASCII( str )` | Devuelve el valor ASCII del caracter str. |
| `BIT_LENGTH( str )` | Devuelve la longitu de str en bits (asumiendo caracteres de 8 bits) |
| `CHAR( int )` | Devuelve el caracter correspondiente al valor ASCII de int. |
| `CHAR_LENGTH( str )` | Devuelve la longitude de str en caracteres. |
| `CHARACTER_LENGTH( str )` | Devuelve la longitude en caracteres. |
| `CONCAT( str1, str2 )` | Devuelve una cadena que es el resultado de sumar str2 a str1. |
| `INSERT( str1, start, len, str2 )` | Devuelve una caden donde str2 dentro de str1 comenzando en start, reemplazando len caracteres. |
| `LCASE( str )` | Devuelve una cadena con todos los carateres de str en minúculas. |
| `LEFT( str, count )` | Devuelve la parte izquierda de str hasta count caracteres. |
| `LENGTH( str )` | Devuelve el número de caracteres en str excluyendo espacios en blanco. |
| `LOCATE( str1, str2[, start] )` | Devuelve la posición de str1 en str2 con un valor opcional de inicio establecido en start. Si str1 no se encuentra dentro de str2, se devuelve 0. |
| `LTRIM( str )` | Devuelve una cadena con los espacios en blanco del lado izquierdo eliminados. |
| `OCTET_LENGTH( str )` | Devuelve la longitud de la codena en octetos (bytes). |

| | |
|---|---|
| `POSITION( str1 IN str2 )` | Devuelve la posición de str1 dentro de str2. |
| `REPEAT( str, cnt )` | Devuelve una cadena consistende de str repetido cnt veces. |
| `REPLACE( str1, str2, str3 )` | Reemplaza todas las ocurrencias de str2 dentro de str1 con str3. |
| `RIGHT( str, count )` | Devuelve una cadena de count caracteres a la derecha de str. |
| `RTRIM( str )` | Devuelve una cadena con los espaciois en blanco de la derecha eliminados. |
| `SPACE( count )` | Devuelve una cadena de espacios en blanco de longitud count. |
| `SUBSTRING( str, pos, len )` | Devuelve una proción de str comenzando en pos y de longitude len. |
| `UCASE( str )` | Devuelve una caden con todos los caracteres convertidos a mayúsculas. |
| `UPPER (str)` | Devuelve a una cadena con los caracteres en minúscula convertidos a mayúscula. |

**Funciones de Fecha / Hora:**

| | |
|---|---|
| `CURDATE( )` | Devuelve la fecha del dia actual |
| `CURRENT_DATE( )` | Igual que CURDATE() |
| `CURRENT_TIME([precision])` | Igual a CURTIME pero con precisión de segundos opcional. |
| `CURRENT_TIMESTAMP([precision])` | Devuelve un TIMESTAMP de la hora local con precision de segundos opcional |
| `CURTIME( )` | Devuelve la hora de la computadora. |
| `DAYNAME(date)` | Devuelve el nombre del día de la semana a partir de un parametro CHAR\|DATE\|TIMESTAMP. |
| `DAYOFMONTH(date)` | Devuelve el número de mes a partir de un parámetro CHAR\|DATE\|TIMESTAMP. |
| `DAYOFWEEK(date)` | Devuelve un número de acuerdo al día de la semana donde 1 es Domingo, a partir de un parámetro CHAR\|DATE\|TIMESTAMP. |
| `DAYOFYEAR(date)` | Devuelve el dia del año a partir de un parámetro CHAR\|DATE\|TIMESTAMP. |

`EXTRACT(time-value FROM time-date)` Extrae el año, mes, dia, hora, minuto o segundo de un valor TIMESTAMP

`HOUR( time )` Devuelve la hora a partir de un valor CHAR|TIME|TIMESTAMP.

`MINUTE( time )` Devuelve el minuto a partir de un valor CHAR|TIME|TIMESTAMP.

`MONTH( date )` Devuelve el mes a partir de un valor CHAR|DATE|TIMESTAMP.

`MONTHNAME( date )` Devuelve el nombre del mes a partir de un valor CHAR|DATE|TIMESTAMP.

`NOW( )` Devuelve un TIMESTAMP de la fecha y hora actuales.

`QUARTER( date )` Devuelve un valor del 1 al 4 dependiendo en que cuatrimestre se le pase como parametro de valor CHAR|DATE|TIMESTAMP.

`SECOND( time )` Devuelve los segundos a partir de un valor CHAR|TIME|TIMESTAMP.

`TIMESTAMPADD(interval,int,timestamp)` Devuelve un TIMESTAMP calculado agregando int intervalos a un TIMESTAMP, los valores de intervalo váidos son:

SQL_TSI_SECOND,
SQL_TSI_MINUTE,
SQL_TSI_HOUR,
SQL_TSI_DAY,
SQL_TSI_WEEK,
SQL_TSI_MONTH,
SQL_TSI_QUARTER,
SQL_TSI_YEAR.

`TIMESTAMPDIFF(interval,timestamp1,timestamp2)` Devuelve el numero de intervalo resultantes de la resta de 2 valores TIMESTAMP Los valores de intervalo soportados son:

SQL_TSI_SECOND,
SQL_TSI_MINUTE,
SQL_TSI_HOUR,
SQL_TSI_DAY,
SQL_TSI_WEEK,
SQL_TSI_MONTH,
SQL_TSI_QUARTER,
SQL_TSI_YEAR.

`WEEK( date )` Devuelve el número de la semana a partir de un valor CHAR|DATE|TIMESTAMP.

`YEAR( date )` Devuelve el año a partir de un valor CHAR|DATE|TIMESTAMP.

**Otras funciones:**

`CONVERT(expr,data-type)` Devuelve el valor de expr convertido a data-type, los parametros de

data-type soportados son:
SQL_BINARY,
SQL_VARBINARY,
SQL_BIN,
SQL_VARCHAR,
SQL_CHAR,
SQL_DATE,
SQL_DOUBLE,
SQL_INTEGER,
SQL_NUMERIC,
SQL_TIME,
SQL_TIMESTAMP.

| | |
|---|---|
| `DATABASE( )` | Devuelve el nmbre de la base de datos en la conexión |
| `DIFFERENCE(str1,str2)` | Devuelve un valor que indica la diferencia entre los valores retornados por la function SOUNDEX() para str1 y str2. El Valor devuelto se encuentra en el rango de 0 y 4. El valor 4 indica mayor acercamiento fonetico, mientras que el valor de 0 indica acercamiento nulo. |
| `SOUNDEX( str )` | Esta function devuelve una codificación fonética de 4 dígitos de una cadena str. La codificación lleva la forma <letra><digito><digito><digito>. |
| `IFNULL( expr, value )` | Si expr es NULL, devuelve value. Si exp. No es NULL se devuelve expr. |
| `ISNULL( expr, value )` | Si expr es NULL, devuelve value. Si expr no es NULL devuelve expr. |
| `USER( )` | Devuelve el nombre del usuario en la base de datos. |
| `IIF(boolean_expr,true_expr,false_expr)` | Si Boolean_expr es verdadeor, se devuelve la evaluación de true_expr, si no, se devuelve la evaluación de false_expr |


**Cibernética y Tecnología S.A. de C.V.**
**Ave. Valle de Toluca No. 25 1er Piso Oficina 1**
**Fracc. El Mirador**
**Naucalpan, Edo. De México**
**530350**

**Visite nuestro sitio Web: www.ciber-tec.com**

**O envienos un correo electrónico: info@ciber-tec.com**