Manipulación de datos (DML: Data Manipulation Language)

SELECT [*predicado*] { * | *tabla*.* | [*tabla.*]*campo1* [AS *alias1*] [, [*tabla*.]*campo2* [AS *alias2*] [, ...]] } FROM *expresióntabla* [WHERE ...] [GROUP BY ...] [HAVING ...] [ORDER BY ...] [WITH OWNERACCESS OPTION]

Permite seleccionar algunos campos de algunos registros procedentes de una tabla, consulta o combinación de tablas y consultas. Cada cláusula significa lo siguientes:

- *predicado* puede ser ALL, DISTINCT, DISTINCTROW o bien TOP *n* [PERCENT], e indica qué registros deben tomarse, de entre los que cumplan las condiciones impuestas a los mismo. Si se omite, equivale a especificar ALL, que implica tomar todos los registros.
- * indica tomar todos los campos de las tablas especificadas en la cláusula FROM.
- *tabla*.* indica tomar todos los campos de la tabla (o consulta) *tabla*, debiendo aparecer ésta en la cláusula FROM.
- [*tabla*.]*campo* [AS *alias*] indica que se debe tomar el campo *campo* de entre los que aparezcan en las tablas o consultas de la cláusula FROM. Si existen campos con nombre iguales en tablas distintas, de indicará cuál de ellos de desea tomar anteponiéndole el nombre de su *tabla*. Si se desea que en la consulta resultante el *campo* aparezca con otro nombre, se usará AS *alias*, siendo éste el nuevo nombre que se desea dar al campo. Adviértase que además de *campo* puede especificarse cualquier expresión SQL válida que se desee, así como que *, *tabla*.* y [*tabla*.]*campo* [AS *alias*] pueden combinarse entre sí mediante comas para indicar más campos que se deseen recuperar.
- FROM *expresióntabla* indica de qué tabla, consulta o combinación de tablas o consultas quieren tomarse los registros.
- WHERE ... indica una condición opcional a aplicar sobre los registros, de forma que sólo se recuperen los que verifiquen dicha condición. Si se omite, equivale a especificar WHERE TRUE; es decir, se toman todos los registros que proceden de la cláusula FROM.
- GROUP BY ... indica por qué campos quiere realizarse una agrupación.
- HAVING ... indica una condición opcional a aplicar sobre los campos agrupados o los cálculos agregados realizados en una consulta sobre la que se realizó una agrupación (GROUP BY). Si se omite, equivale a especificar HAVING TRUE; es decir, se toman todos los registros que resulten de la agrupación.
- ORDER BY ... indica el campo o campos por los que se desean ordenar los registros, de forma que éstos se devuelvan ordenados. Si se omite, los registros se recuperarán en un orden arbitrario, no determinado.
- WITH OWNERACCESS OPTION indica que se desea dar al usuario de la consulta los mismos permisos que el usuario que la creó. Sólo es útil en entornos multiusuario en los que se han establecido grupos de seguridad.

Ejemplos:

- SELECT * FROM Empleados;
- SELECT Apellidos, Nombre FROM Empleados WHERE Departamento = 'Ventas';
- SELECT FechaNac AS Nacimiento FROM Empleados ORDER BY Apellidos, Nombre;

El *predicado* puede tomar los valores ALL, DISTINCT, DISTINCTROW o TOP *n* [PERCENT]. Cada una de estas opciones significa lo siguiente:

- ALL: Se recuperan todos los registros (que verifiquen las condiciones WHERE y HAVING, si éstas existen).
- DISTINCT: Se recuperan todos los registros salvo aquellos que presenten valores duplicados en los campos seleccionados (en la cláusula SELECT).
- DISTINCTROW: Se recuperan todos los registros salvo aquellos que presenten valores duplicados en todos los campos, y no sólo en los seleccionados (en la cláusula SELECT).
- TOP *n*: Recupera los *n* primeros registros, según el orden especificado en la consulta (ORDER BY). Si no se especificó un orden, se toman *n* registros arbitrariamente.
- TOP *n* PERCENT: Igual que al anterior, pero ahora *n* representa un porcentaje respecto al total de registros.

Ejemplos:

- SELECT ALL * FROM Empleados ORDER BY EmpleadoId;
- SELECT DISTINCT Apellidos FROM Empleados;
- SELECT TOP 25 Apellidos, Nombre FROM Estudiantes WHERE AñoGraduacion = 1994 ORDER BY NotaMedia DESC;

La cláusula FROM especifica las tablas o consultas que contienen los registros que se desean recuperar. Puede presentar alguna de las siguientes formas:

- FROM { *tabla* | *consulta* }: Recupera los registros de la *tabla* o *consulta* especificada.
- FROM *tabla1*, *tabla2*: Calcula el producto cartesiano de la *tabla1* y la *tabla2* (que también pueden ser consultas). El producto cartesiano consta de todas las posibles combinaciones de un registro de la *tabla1* y otro de la *tabla2*.

- FROM *tabla1* { INNER | LEFT | RIGHT } JOIN *tabla2* ON *tabla1*.*campo1* = *tabla2*.*campo2*: Relaciona los registros de la *tabla1* con los de la *tabla2* (que también pueden ser consultas) de forma similar al producto cartesiano, pero devolviendo sólo aquellas parejas en las que se verifique la relación *tabla1*.*campo1* = *tabla2*.*campo2*. Si se especifica INNER JOIN, sólo se devuelven las parejas que verifiquen la relación. Si se especifica LEFT JOIN, se devuelven todos los registros de *tabla1* (que está la izquierda del JOIN), relacionados con los registros de *tabla2* cuando alguno verifique la relación. Si se especifica RIGHT JOIN, se devuelven los todos los registros de *tabla2*, relacionados cuando proceda con los registros de *tabla1*. En la relación ON, no sólo se permite la igualdad (=), sino también las comparaciones <, >, <=, >= y <>. Además, pueden relacionarse dos tablas mediante más de dos campos, indicando cómo se combinan las relaciones entre sí mediante AND, OR, etcétera.

Las anteriores combinaciones de tablas y consultas pueden anidarse entre sí tanto como se desee (usando paréntesis para indicar su precedencia), pero deben observarse las siguientes reglas:

- No pueden relacionarse mediante JOIN dos productos cartesianos, pero sí puede realizarse el producto cartesiano de otros productos cartesianos o relaciones JOIN.
- Puede anidarse un INNER JOIN dentro de un LEFT JOIN o un RIGHT JOIN, pero no a la inversa.

Puede indicarse además que una tabla o consulta reside en una base de datos externa usando la sintaxis:

- FROM *tabla* IN { "*ruta*" | "*ruta*" "*tipo*" | "" **[***conexión***]** }

Adviértase que los corchetes en negrita indica que deben escribirse corchetes en esa posición de la cláusula. Para esta última forma, además, una alternativa más cómoda es: FROM **[***conexión***]**.*tabla*.

Ejemplos:

- SELECT * FROM Categorias, Empleados WHERE Categorias.IdCategoria = Empleados.CategoriaId
- SELECT Categorias.Nombre AS NombreCategoria, Productos.Nombre AS NombreProducto FROM Categorias INNER JOIN Productos ON Categorias.IdCategoria = Productos.CategoriaId;
- SELECT DISTINCTROW Sum(PrecioUd * Cantidad) AS Ventas, Nombre & ' ' & Apellidos AS Name FROM Empleados INNER JOIN (Pedidos INNER JOIN [Detalle Pedidos] ON [Detalle Pedidos].PedidoId = Pedidos.IdPedido) ON Pedidos.EmpleadoId = Empleados.IdEmpleado GROUP BY Nombre & ' ' & Apellidos AS Name;
- SELECT Categorias.Nombre, Productos.Nombre FROM Categorias LEFT JOIN Productos ON Categorias.IdCategoria = Products.CategoriaId;
- SELECT Departamentos.Nombre, Apellidos, Empleados.Name FROM Departmentos RIGHT JOIN Empleados ON Departamentos.IdDepartamento = Empleados.DepartamentoId ORDER BY Departamentos.Nombre;
- SELECT ClienteId FROM Clientes IN OtraBD.mdb WHERE Nombre LIKE 'A*';
- SELECT ClienteId FROM Clientes IN "C:\DBASE\VENTAS" "dBASE IV;" WHERE Nombre LIKE 'A*';
- SELECT ClienteId FROM Clientes IN "" [Paradox 4.x;Database=C:\DATOS\VENTAS;] WHERE Nombre LIKE 'A*';
- SELECT ClienteId FROM [EXCEL 8.0;Database=c:\doc\mihoja.xls;].RangoClientes WHERE Nombre LIKE 'A*';

La cláusula WHERE permite especificar una condición, de forma que la consulta sólo tomará aquellos registros procedentes de la cláusula FROM que hagan cierta dicha condición. Omitirla equivale a especificar WHERE TRUE que, por cumplirse siempre, hace que se tomen todos los registros. La condición puede ser tan compleja como queramos, y puede hacer referencia a los campos de origen, a los alias de los mismos, a constantes, parámetros y todas las funciones (de sistema y usuario) que se desee.

Ejemplos:

- SELECT Apellidos, Nombre, Sueldo FROM Empleados WHERE Sueldo > 2500000;
- SELECT * FROM Pedidos WHERE FechaSalida = #5/10/96#;
- SELECT * FROM Pedidos WHERE FechaSalida = DateValue('10/5/96');
- SELECT * FROM Clientes WHERE CP BETWEEN '28001' AND '28045';
- SELECT * FROM Pedidos WHERE Provincia IN ('Madrid', 'Sevilla');
- SELECT * FROM Clientes WHERE Apellidos LIKE '[BZ]*';
- SELECT * FROM Clientes WHERE CP IS NULL AND Provincia = 'Madrid';

La cláusula GROUP BY permite realizar agrupaciones y cálculos agregados sobre los registros de una consulta. Al especificarla, debe ir seguida de la lista de campos por los que se desea agrupar. Los campos que podrán recuperarse en la cláusula SELECT serán, además de todos aquellos por los que se agrupen los registros, cualquier campo calculado agregado (es decir, aquellos que calculan un valor a partir de campos de los registros agrupados mediante las funciones de agregación COUNT, SUM, MIN, MAX, AVG, VAR, VARP, STDEV y STDEVP) o constante. Si una consulta sólo recupera campos agregados que se calculan sobre todos los registros, puede omitirse el GROUP BY.

Ejemplos:

- SELECT Nombre, Count(Nombre) FROM Empleados GROUP BY Nombre;
- SELECT Nombre, Count(Nombre) FROM Empleados WHERE Sueldo > 2000000 GROUP BY Nombre;

- SELECT Count(*) FROM Empleados;
- SELECT CategoriaId, Min(Sueldo), Max(Sueldo) AS FROM Empleados GROUP BY CategoriaId;
- SELECT Avg(Sueldo) FROM Empleados WHERE CategoriaId = 1;

La cláusula HAVING permite establecer una condición sobre los resultados resultantes de una agrupación establecida en una consulta mediante GROUP BY. Es idéntica a la cláusula WHERE, salvo porque mientras que en ésta se hace referencia a campos de los registros individuales recuperados de las tablas indicadas en la cláusula FROM, en la cláusula HAVING se hace referencia a los campos por los que se ha agrupado o a cálculos agregados.
Ejemplo:

- SELECT CategoriaId, Sum(Stock) FROM Productos GROUP BY CategoriaId HAVING Sum(Stock) > 100;
- SELECT CategoriaId, Count(CategoriaId) FROM Empleados GROUP BY CategoriaId HAVING Avg(Sueldo) > 1000000;

La cláusula ORDER BY permite establecer en qué orden se devolverán los registros seleccionados por la consulta. Su sintaxis es:

- ORDER BY *campo1* [ASC | DESC] [, *campo2* [ASC | DESC] [, …]]

Los *campos* que aparecen en ORDER BY pueden ser cualquiera de los recuperados en la cláusula SELECT (incluyendo los alias que se hayan definido), o cualquier expresión SQL válida, o un ordinal que indicará la posición del campo (en la cláusula SELECT) por el que se desee ordenar. Si la consulta es agrupada (GROUP BY), sólo podrán aparecer en esta cláusula campos con las mismas restricciones que para HAVING. Con ASC y DESC se indica si se desea que la ordenación se realice ascendente o descendentemente; de omitirse, se supone ASC.
Ejemplos:

- SELECT Apellidos, Nombre FROM Empleados ORDER BY Apellidos;
- SELECT Apellidos, Nombre, Sueldo FROM Empleados ORDER BY Sueldo DESC, Apellidos;

**SELECT [*predicado*] { * | *tabla*.* | [*tabla.*]*campo1* [AS *alias1*] [, [*tabla*.]*campo2* [AS *alias2*] [, ...]] }
INTO *destino* FROM *expresióntabla* [IN *basededatosexterna*] [WHERE ...] [GROUP BY ...] [HAVING ...] [ORDER BY ...]**

Esta sentencia permite seleccionar unos campos determinados de unos registros dados y crear una nueva tabla *destino* en la que insertará dichos campos. Esta es la única diferencia con el SELECT convencional (que en lugar de crear una tabla e insertar los registros, los devuelve). Para determinar los tipos y nombres de los campos de la tabla destino se usan los de los campos especificados en la cláusula SELECT, por lo que es importante dar nombre válidos a todos los campos que lo necesiten mediante AS. Si en lugar de crear una nueva tabla se desea añadir el resultado de la consulta a una tabla ya existente, debe usarse la sentencia INSERT INTO.
Ejemplos:

- SELECT * INTO CopiaEmpleados FROM Empleados;
- SELECT CategoriaId, Avg(Sueldo) AS SueldoMedio INTO Salario FROM Empleados GROUP BY CategoriaId;

[TABLE] *consulta1* UNION [ALL] [TABLE] *consulta2* [UNION [ALL] *consulta3* [...]]

Crea una consulta de unión, que combina el resultado de dos o más consultas o tablas independientes en una sola. Si se especifica UNION, se devuelven los registros de las dos consultas implicadas, eliminando aquellos que resulten ser idénticos. Si se especifica UNION ALL, no se omiten estos últimos (y la consulta de unión resulta mucho más rápida). Todas las *consutas* deben tener el mismo número de campos, pero no se exigen que éstos tengan el mismo tamaño ni el mismo tipo de dato. Para obtener los nombres de los campos del resultado final sólo se examina la primera consulta (*consulta1*), por lo que sólo es necesario usar alias en ésta. Pueden usarse cláusulas GROUP BY y HAVING en cada una de las consultas, pero sólo puede usarse una cláusula ORDER BY al final, para indicar cómo se ordena la unión de todos los registros. Este ORDER BY hace referencia a los campos de la forma en que se les llamó en el primer SELECT.
Ejemplos:

- TABLE NuevosClientes UNION ALL SELECT * FROM Clientes WHERE TotalPedidos > 1000;
- SELECT Nombre, Ciudad FROM Proveedores WHERE Pais = 'Brazil' UNION SELECT Nombre, Ciudad FROM Clientes WHERE Pais = 'Brazil';

INSERT INTO *destino* [(*campo1* [, *campo2* [, ...]])] VALUES (*valor1* [, *valor2* [, ...]])

Añade un nuevo registro al *destino*, que puede ser una tabla o consulta de la base de datos sobre la que estemos trabajando. En la sentencia SQL se indican los *campos* a los que se asignan los *valores* deseados; si se omiten los nombre de los *campos*, se tomarán éstos en orden de la definición del *destino* y será obligatorio especificar

valores para todos los campos existentes. Hay que especificar cada uno de los campos del registro para los que se desea asignar un valor, así como dichos valores. Para aquellos campos de la tabla *destino* que se omitan, se insertará el valor por defecto o NULL. El registro se añade al final de la tabla.
Ejemplo:

- INSERT INTO Empleados (Nombre, Apellidos, Cargo) VALUES ('Antonio', 'Benítez Gil', 'Ingeniero');

INSERT INTO *destino* [(*campo1* [, *campo2* [, ...]])] [IN *basededatosexterna*] SELECT *campo1* [, *campo2* [, ...]] FROM *origen*

Selecciona algunos *campos* de una consulta o tabla *origen* y los inserta en la tabla o consulta *destino*. Si especifica una lista de *campos* tras la cláusula INSERT INTO, éstos indican los campos del *destino* en el que se insertarán los datos recuperados del *origen*; si se omiten, se tomarán los nombres de los campos recuperados del *origen* y se insertarán en las columnas homónimas del *destino*. Adviértase que *origen* es una expresión de consulta cualquiera, idéntica a las que aparecen en las sentencias SELECT; es decir, puede especificarse una cláusula WHERE, una agrupación, ordenación, etcétera. Opcionalmente, puede usarse la cláusula IN para indicar que el *destino* reside a una base de datos externa (para más detalles, véase la definición de SELECT).
Ejemplo:

- INSERT INTO Clientes SELECT * FROM NuevosClientes WHERE FechaAlta > #1/1/2000#

UPDATE *origen* SET *valores* [WHERE *condición*]

Permite actualizar los campos que deseemos de un conjunto de registros determinado que cumpla una condición dada. Las cláusulas de esta sentencia son:

- UPDATE *origen*: Indica el origen de los registros que se desean actualizar. Puede ser una tabla, una consulta o una combinación de tablas y/o consulta, de la misma forma que aparecen en la cláusula FROM de la sentencia SELECT (véase la definición de ésta para más detalles).
- SET *valores*: Indica qué campos se quieren actualizar y cómo. Es una lista de asignaciones, separadas entre sí por comas, con la forma: *campo* = *valor*, donde *campo* es cualquiera de los campos del *origen* y *valor* es el resultado de cualquier expresión SQL válida, donde pueden aparecer los campos del *origen*, incluso recursivamente.
- WHERE *condición*: Indica qué *condición* han de verificar los registros del *origen* para ser actualizados. Los que no la cumplan no serán modificados. Si se omite esta cláusula, se actualizarán todos los registros del *origen*. Esta cláusula es idéntica al WHERE de la sentencia SELECT (véase la definición de ésta para más detalles).

Adviértase que, en general, no es posible deshacer los cambios realizados tras un UPDATE.
Ejemplos:

- UPDATE Pedidos SET GastosEnvio = GastosEnvio * 1.25, Impuestos = 0.0 WHERE Destino = 'Canarias';
- UPDATE Empleados SET Sueldo = 5000000 WHERE Sueldo = 4750000;

DELETE [[*tabla*.]*] FROM *origen* [WHERE *condición*]

Elimina los registros que cumplan una *condición* de los pertenecientes a un *origen* de datos dado. Adviértase que *origen* puede ser una tabla, consulta o combinación de tablas y/o consultas cualquiera. Si dicho *origen* involucra más de una tabla, es posible indicar en la cláusula DELETE la tabla cuyos registros quieren eliminarse. No es posible, sin embargo, eliminar sólo algunas columnas de dicha tabla: o se borra el registro completo, o no se borra nada. Puede especificarse además una *condición* que deben verificar los registros que desean borrarse del *origen*; si dicha condición se omiten, se borrarán todos los registros. Si se estableció alguna relación de integridad referencial entre tablas, el DELETE puede provocar borrados en cascada. Téngase en cuenta que, en general, no es posible recuperar los registros eliminados tras un DELETE, y que aunque eliminemos todos los registros de una tabla, ésta seguirá existiendo (aunque vacía) en la base de datos: para eliminar también la definición de la tabla, debe usarse una sentencia DROP TABLE (véase más adelante).
Ejemplo:

- DELETE * FROM Empleados WHERE Cargo = 'Contable';

Definición de datos (DDL: Data Definition Language)

CREATE TABLE *tabla* (*campo1 tipo*[(*tamaño*)] [NOT NULL] [*índice1*] [, *campo2 tipo*[(*tamaño*)] [NOT NULL] [*índice2*] [, ...]] [, CONSTRAINT *índicemúltiple* [, ...]])

Esta sentencia permite crear una *tabla* con los campos indicados, especificando el *tipo* y (opcionalmente, si éste lo requiere) el *tamaño* de los mismos. Si para algún campo de indica NOT NULL, éste no podrá tomar valores nulos. Se permite establecer índice simples (*índice1, índice2, …*) sobre cada campo. Un índice simple tiene la forma:

CONSTRAINT *nombre* { PRIMARY KEY | UNIQUE | NOT NULL | REFERENCES *tablaajena* [(*campoajeno*)] }

Esta cláusula crea un índice simple llamado *nombre*. El significado de cada opción es:

- PRIMARY KEY: Indica que el campo sobre el que se define es clave primaria.
- UNIQUE: Indica que el campo sobre el que se define no acepta valores duplicados.
- NOT NULL: Indica que el campo sobre el que se define no acepta valores nulos.
- REFERENCES …: Establece una relación con una *tablaajena*. Si se especifica el *campoajeno*, se establece la relación entre éste (que debe tener definido un índice UNIQUE o PRIMARY KEY) y el campo sobre el que se define la referencia. Si se omite, se establece entre la clave primaria de la *tablaajena* y el campo sobre el que se define la referencia.

También se permite definir índices múltiples, que tienen la siguiente forma:

CONSTRAINT *nombre* { PRIMARY KEY (*campo1*[, *campo2* [, ...]]) |
UNIQUE (*campo1*[, *campo2* [, ...]]) | NOT NULL (*campo1*[, *campo2* [, ...]]) |
FOREIGN KEY (*campo1*[, *campo2* [, ...]]) REFERENCES *tablaajena* [(*campoajeno1* [, *campoajeno2* [, ...]])]}

Esta cláusula crea un índice múltiple llamado *nombre* definido sobre los campos *campo1, campo2, …*. El significado de cada opción es:

- PRIMARY KEY: Los campos del índice múltiple son la clave primaria de la tabla.
- UNIQUE: Los campos del índice múltiple no pueden tomar valores repetidos (conjuntamente).
- NOT NULL: Los campos del índice múltiple no pueden ser nulos.
- FOREIGN KEY … REFERENCES …: Establece una relación con una *tablaajena*.

Ejemplos:

- CREATE TABLE EstaTabla (Nombre TEXT, Apellido TEXT);
- CREATE TABLE MiTabla (Nombre TEXT, Apellido TEXT, FechaNac DATETIME, CONSTRAINT MiIndice UNIQUE (Nombre, Apellido, FechaNac));
- CREATE TABLE NuevaTabla (Nombre TEXT, Apellido TEXT, Clave INTEGER CONSTRAINT MiIndice PRIMARY KEY);

ALTER TABLE *tabla* {ADD {COLUMN *campo tipo*[(*tamaño*)] [NOT NULL] [CONSTRAINT *índice...*] | CONSTRAINT *índicemúltiple*} | DROP {COLUMN *campo* | CONSTRAINT *índice*} }

- ALTER TABLE *tabla* ADD COLUMN *campo tipo*[(*tamaño*)] [NOT NULL] [CONSTRAINT *índice*]

  Añade una columna llamada *campo* del *tipo* y (opcionalmente, si éste lo requiere) *tamaño* especificados. Si se especifica NOT NULL, entonces el *campo* no podrá tomar valores nulos. Opcionalmente, puede especificarse un índice un restricción sobre dicho campo (para más información, véase la definición de CREATE TABLE).

- ALTER TABLE *tabla* ADD CONSTRAINT *índice* { PRIMARY KEY | UNIQUE | NOT NULL | REFERENCES *tablaajena* [(*campoajeno1, campoajeno2, …*)] }

  Añade un índice múltiple a la *tabla*. Para más detalles, véase la definición de CREATE TABLE.

- ALTER TABLE *tabla* DROP COLUMN *campo*

  Elimina el *campo* de la *tabla*, y por tanto todos los datos que contuviera dicha columna.

- ALTER TABLE *tabla* DROP CONSTRAINT *índice*

  Elimina el *índice* (simple o múltiple) previamente definido sobre *tabla*. No modifica los datos de la misma.

Ejemplos:

- ALTER TABLE Empleados ADD COLUMN Sueldo CURRENCY;
- ALTER TABLE Empleados DROP COLUMN Sueldo;
- ALTER TABLE Pedidos ADD CONSTRAINT RelPedidos FOREIGN KEY (EmpleadoNum) REFERENCES Empleados (NumEmpleado);
- ALTER TABLE Pedidos DROP CONSTRAINT RelPedidos;

CREATE [UNIQUE] INDEX *índice* ON *tabla* (*campo1* [ASC|DESC][, *campo2* [ASC|DESC], ...]) [WITH { PRIMARY | DISALLOW NULL | IGNORE NULL }]

Crea un índice sobre una *tabla* para permitir una rápida ordenación por los campos indicados. Para cada uno de éstos, se indica si deben ordenarse ascendente (ASC) o descendentemente (DESC); si omite, se asume que se desea ordenar ascendentemente. Si se especifica que el índice es UNIQUE, entonces no se permitirán valores duplicados en los campos del mismo. Además, puede opcionalmente indicarse que:

- PRIMARY: El índice constituya la clave primaria de la *tabla*.
- DISALLOW NULL: Los campos que forman el índice no acepten nulos.
- IGNORE NULL: Los registros con nulos en los campos que forman el índice no se incluyan en el mismo.

Ejemplos:

- CREATE INDEX NuevoIndice ON Empleados (Telefono, CP);
- CREATE UNIQUE INDEX Id ON Clientes (IdCliente) WITH DISALLOW NULL;

DROP { TABLE *tabla* | INDEX *índice* ON *tabla* }

Permite eliminar una *tabla* (TABLE) o un *índice* simple o múltiple definido sobre una *tabla* (INDEX … ON).
Ejemplos:

- DROP INDEX NuevoIndice ON Empleados;
- DROP TABLE Empleados;

Tipos de datos

En la siguiente tabla se muestran los tipos de datos ANSI SQL (que deben estar presentes en todas las implementaciones estándar de SQL) y las equivalencias de los mismos con los tipos de datos Microsoft Jet SQL, así como sus sinónimos válidos.

Tipo de dato ANSI SQL	Tipo de dato Microsoft Jet SQL	Sinónimo
BIT, BIT VARYING	BINARY (Ver notas)	VARBINARY
(No soportado)	BIT (Ver notas)	BOOLEAN, LOGICAL, LOGICAL1, YESNO
(No soportado)	BYTE	INTEGER1
(No soportado)	COUNTER	AUTOINCREMENT
(No soportado)	CURRENCY	MONEY
DATE, TIME, TIMESTAMP	DATETIME	DATE, TIME, TIMESTAMP
(No soportado)	GUID	
DECIMAL	(No soportado)	
REAL	SINGLE	FLOAT4, IEEESINGLE, REAL
DOUBLE PRECISION, FLOAT	DOUBLE	FLOAT, FLOAT8, IEEEDOUBLE, NUMBER, NUMERIC
SMALLINT	SHORT	INTEGER2, SMALLINT
INTEGER	LONG	INT, INTEGER, INTEGER4
INTERVAL	(No soportado)	
(No soportado)	LONGBINARY	GENERAL, OLEOBJECT
(No soportado)	LONGTEXT	LONGCHAR, MEMO, NOTE
CHARACTER, CHARACTER VARYING	TEXT	ALPHANUMERIC, CHAR, CHARACTER, STRING, VARCHAR
(No soportado)	VALUE (Ver notas)	

Notas:

- El tipo de dato ANSI SQL **BIT** no se corresponde con el tipo de dato Microsoft Jet SQL **BIT**, sino con el tipo de dato **BINARY**. No hay equivalente en ANSI SQL para el tipo de dato Microsoft Jet SQL **BIT**.
- La palabra reservada **VALUE** no representa ningún tipo de dato definido en Microsoft Jet SQL, pero sí en otros dialectos de SQL.

La tabla siguiente muestra los tipos de datos primarios definidos en Microsoft Jet SQL, así como sus correspondientes tamaño y significado. Los sinónimos permitidos para cada uno de ellos aparecen en la tabla anterior.

Tipo de dato	Tamaño	Descripción
BINARY	1 byte por carácter	Cualquier tipo de dato puede almacenarse en un campo de este tipo. No se hace ningún tipo de traducción (por ejemplo, a texto) a los datos. Éstos se recuperarán exactamente igual a como se introdujeron.
BIT	1 byte	Valores YES y NO y campos que contienen valores lógicos.
BYTE	1 byte	Un valor entero entre 0 y 255.
COUNTER	4 bytes	Un número entero automáticamente incrementado por la base de datos cada vez que se añade un nuevo registro a la tabla. Internamente corresponde a un LONG.
CURRENCY	8 bytes	Un número con 4 decimales comprendido entre -922.337.203.685.477,5808 y 922.337.203.685.477,5807.
DATETIME	8 bytes	Una valor de fecha u hora comprendido entre los años 100 y 9999. Internamente corresponde a un DOUBLE.
GUID	128 bits	Un número de identificación única usado para llamadas remotas.
SINGLE	4 bytes	Un número en coma flotante de simple precisión en los rangos -3,402823E38 a -1,401298E-45 para valores negativos, 1,401298E-45 a 3,402823E38 para valores positivos, y el 0.
DOUBLE	8 bytes	Un número en coma flotante de doble precisión en los rangos -1,79769313486232E308 a -4,94065645841247E-324 para valores negativos, 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos, y el 0.
SHORT	2 bytes	Un entero corto comprendido entre -32.768 y 32.767.
LONG	4 bytes	Un entero largo comprendido entre -2.147.483.648 y 2.147.483.647.
LONGTEXT	1 byte por carácter	Texto de 0 a 1,2 gigabytes.
LONGBINARY	Según se requiera	Datos de 0 a 1,2 gigabytes. Se usa para almacenar objetos OLE.
TEXT	1 byte por carácter	Texto de 0 a 255 caracteres.