TEMA 5: El lenguaje SQL III: Creación de bases de datos y seguridad.

5.1 Introducción.

Hasta el momento hemos visto sentencias SQL (SELECT, INSERT, etc.), que permiten manipular los datos de SQL. Estas sentencias permiten modificar los datos almacenados en una base de datos, pero no pueden cambiar su estructura. Ninguna de estas sentencias crea o suprime tablas o columnas de una base de datos.

Los cambios en la estructura de la base de datos son manejados por un conjunto de sentencias SQL denominadas como *lenguaje de definición de datos*. Estas sentencias permiten:

- Definir y crear una nueva tabla.
- Suprimir una tabla que ya no se necesita.
- Cambiar la definición de una tabla existente.
- Definir una tabla virtual (o vista) de datos.
- Establecer controles de seguridad para una base de datos.

Desarrollaremos los apartados anteriores en tres apartados distintos, centrándonos en el primero en la creación y modificación de la base de datos, para exponer en el segundo apartado la creación de tablas virtuales (vistas) y en un tercero la seguridad en la base de datos.

5.2 Creación de una base de datos.

En instalaciones para grandes computadoras, la creación de una base de datos suele ser responsabilidad del administrador de la base de datos. Sin embargo, en instalaciones sobre minicomputadoras los usuarios individuales pueden permitirse crear sus propias bases de datos personales. Existen tres sentencias que se emplean en SQL para crear o modificar una base de datos. Estas sentencias son:

- CREATE, que define y crea un objeto en la base de datos.
- DROP, que elimina un objeto existente en la base de datos.
- ALTER, que modifica la definición de un objeto de la base de datos.

La estructura más importante de una base de datos relacional es la tabla. En una base de datos multiusuario, las tablas principales son creadas por el administrador de la base de datos y utilizadas con posterioridad por los usuarios. Sin embargo, estos pueden encontrar conveniente definir tablas propias para almacenar datos personales o extraídos

de otras tablas. En una base de datos sobre un computador personal, la estructura de las tablas puede ser alterarse sin preocuparse del resto de usuarios, pues un mismo usuario es a la vez usuario y administrador de la base de datos.

5.3 Creación de una tabla.

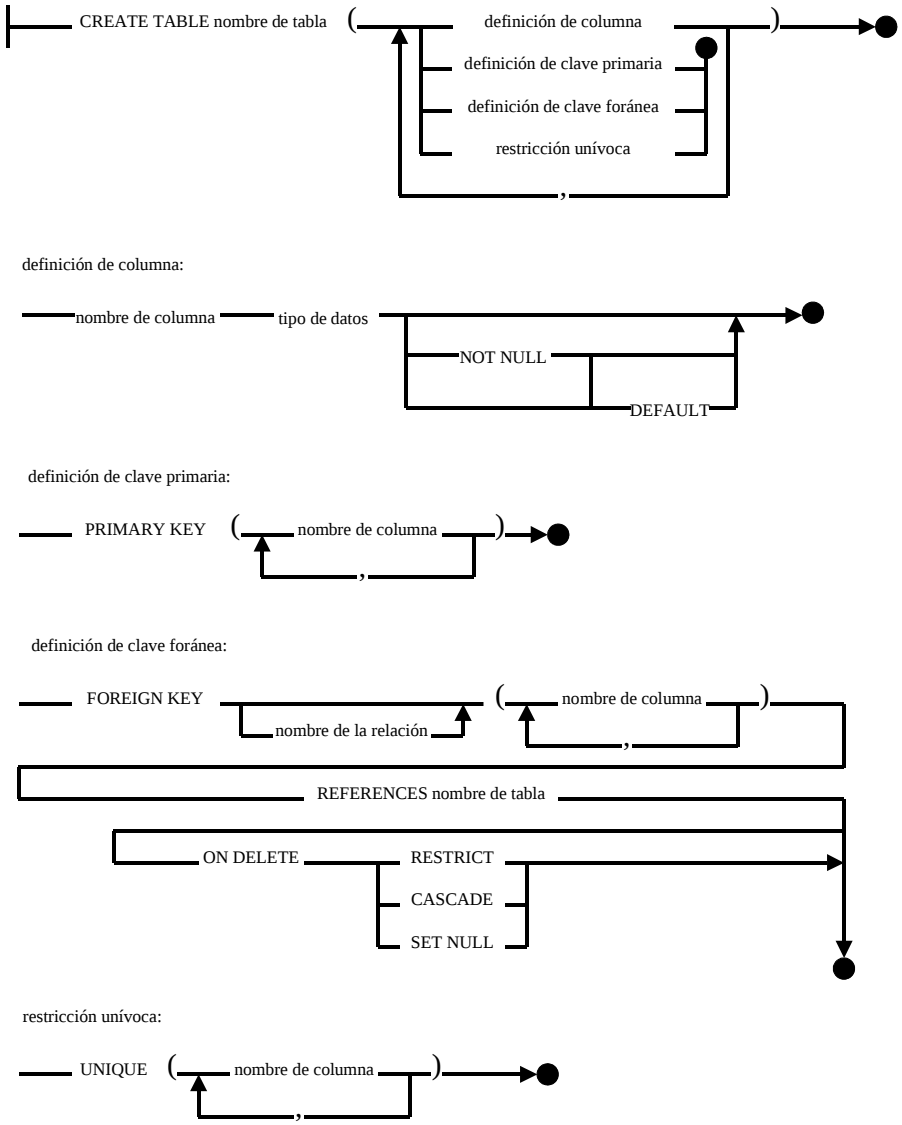

Figura 5.3.1: Diagrama sintáctico de la sentencia CREATE TABLE.

La creación de una tabla se realiza con la sentencia CREATE TABLE, cuyo diagrama se muestra en la figura 5.3.1.

El diagrama sintáctico parece complejo, ya que hay muchas partes de la definición que especificar y muchas opciones para cada elemento. En la práctica, la creación de una nueva tabla es sencilla y dividiremos su estudio en cuatro partes.

5.3.1 Definiciones de columnas.

Las columnas de la tabla recién creada se definen en el cuerpo de la sentencia CREATE TABLE. Las columnas aparecen en una lista separada por comas e insertada

entre paréntesis. El orden de las definiciones de las columnas determina el orden de izquierda a derecha de las columnas en la tabla. Cada definición especifica:

- El nombre de la columna, que se utiliza para referirse a la columna en las sentencias SQL. Cada columna de la tabla debe tener un nombre único, pero los nombres pueden ser iguales a los de las columnas de otras tablas.

- El tipo de datos de la columna, que identifica la clase de datos que la columna almacena.

- Si la columna no puede contener datos nulos, la cláusula NOT NULL impide que aparezcan valores NULL en la columna, en caso contrario se permiten los valores nulos en la columna.

- Un valor por omisión es opcional para la columna. Se utiliza este valor cuando una sentencia INSERT aplicada a la tabla no especifica un valor para la columna.

Algunos ejemplos de sentencias CREATE TABLE son:

```
CREATE TABLE oficinas
        (oficina INTEGER NOT NULL,
        ciudad VARCHAR(15) NOT NULL,
        region VARCHAR(10) NOT NULL,
        objetivo INTEGER,
        ventas INTEGER NOT NULL)
```

OFICINA	CIUDAD	REGION	OBJETIVO	VENTAS

Tabla 5.3.1.1: Definición de la tabla OFICINAS y sus columnas.

```
CREATE TABLE pedidos
        (num_pedido INTEGER NOT NULL,
        fecha_pedido DATE NOT NULL,
        clie INTEGER NOT NULL,
        rep INTEGER
        fab CHAR(3) NOT NULL,
        producto CHAR(5) NOT NULL,
        cant INTEGER NOT NULL,
        importe INTEGER NOT NULL)
```

NUM_PEDIDO	FECHA_PEDIDO	CLIE	REP	FAB	PRODUCTO	CANT	IMPORTE

Tabla 5.3.1.2: Definición de la tabla PEDIDOS y sus columnas.

5.3.2 Valores por omisión.

El estándar ANSI/ISO permite especificar un valor por omisión para cada columna. Por ejemplo, la siguiente sentencia especifica valores por omisión para la tabla OFICINAS.

```
CREATE TABLE oficinas
```

```
(oficina INTEGER NOT NULL,
ciudad VARCHAR(15) NOT NULL,
region VARCHAR(10) NOT NULL DEFAULT 'Este',
objetivo INTEGER DEFAULT NULL,
ventas INTEGER NOT NULL DEFAULT 0)
```

OFICINA	CIUDAD	REGION	OBJETIVO	VENTAS

Tabla 5.3.2.1: Definición de la tabla OFICINAS con valores por omisión

Sin embargo, otros productos como los SQL de IBM no permiten especificar un valor por omisión diferente para cada columna. En su lugar proporcionan un valor por omisión para cada tipo de dato soportado. Así, el valor por omisión para los datos numéricos es 0, para los datos VARCHAR es la cadena vacía, para los datos CHAR es la cadena llena de blancos y para los datos de fecha y hora es la fecha y hora actuales. Por ello, la creación de la tabla anterior, utilizando sintaxis de IBM, sería:

```
CREATE TABLE oficinas
        (oficina INTEGER NOT NULL,
        ciudad VARCHAR(15) NOT NULL,
        region VARCHAR(10) NOT NULL WITH DEFAULT,
        objetivo INTEGER,
        ventas INTEGER NOT NULL WITH DEFAULT)
```

OFICINA	CIUDAD	REGION	OBJETIVO	VENTAS

Tabla 5.3.2.2: Definición de la tabla OFICINAS con valores por omisión (sintaxis IBM).

5.3.3 Definición de clave primaria y foránea.

Además de la definición de las columnas de una tabla, la sentencia CREATE TABLE identifica la clave primaria de la tabla y las relaciones de la tabla con otras tablas de la base de datos. Las cláusulas PRIMARY KEY y FOREIGN KEY manejan estas funciones.

La cláusula PRIMARY KEY especifica la columna o columnas que forman la clave primaria de la tabla. Al especificar una clave primaria, el gestor de la base de datos requiere automáticamente que el valor de la clave primaria sea único en cada fila de la tabla. Además, la definición de columna para todas las columnas que forman la clave primaria debe especificar que la columna es NOT NULL.

La cláusula FOREIGN KEY especifica una clave foránea en la tabla y la relación que crea con otra tabla (padre) de la base de datos. La cláusula especifica:

- La columna o columnas que forman la clave foránea, todas las cuales son columnas de la tabla que está siendo creada.

- La tabla que es referenciada por la clave foránea. Esta es la tabla padre en la relación, la tabla que está siendo definida es el hijo.

- Un nombre opcional para la relación. El nombre no se utiliza en ninguna sentencia SQL, pero puede aparecer en mensajes de error y es necesario si se desea poder suprimir la clave foránea posteriormente.

- Una regla de supresión opcional para la relación (RESTRICT, CASCADE o SET NULL). Si no se especifica la regla de supresión se utiliza por defecto la regla RESTRICT.

Las reglas de supresión especifican qué debe hacer cuando un usuario trate de suprimir una fila de la tabla padre. Si la regla de supresión es RESTRICT se impide suprimir una fila de la tabla padre si la fila tiene algún hijo. La regla de supresión CASCADE indica que si se suprime una fila padre, todas las filas hijo también sean suprimidas, con lo cual la supresión en la tabla padre se propaga a las tablas hijo. Por último, la regla SET NULL indica que cuando una fila padre sea suprimida, los valores de clave foránea de todas las filas hijo deben ser automáticamente puestas a NULL.

Un ejemplo de creación de una tabla con sus claves primarias y foráneas puede verse a continuación:

```
CREATE TABLE pedidos
        (num_pedido INTEGER NOT NULL,
        fecha_pedido DATE NOT NULL,
        clie INTEGER NOT NULL,
        rep INTEGER,
        fab CHAR(3) NOT NULL,
        producto CHAR(5) NOT NULL,
        cant INTEGER NOT NULL,
        importe INTEGER NOT NULL,
        PRIMARY KEY (num_pedido),
        FOREIGN KEY pedidopor(clie) REFERENCES clientes
                ON DELETE CASCADE,
        FOREIGN KEY tomadopor(rep) REFERENCES repventas
                ON DELETE SET NULL,
        FOREIGN KEY fabricadopor(fab,producto) REFERENCES productos
                ON DELETE RESTRICT)
```

NUM_PEDIDO	FECHA_PEDIDO	CLIE	REP	FAB	PRODUCTO	CANT	IMPORTE

Tabla 5.3.3.1: Definición de la tabla PEDIDOS con las claves primaria y foránea.

La figura siguiente muestra las tres relaciones creadas por la sentencia anterior y los nombres que les asigna.

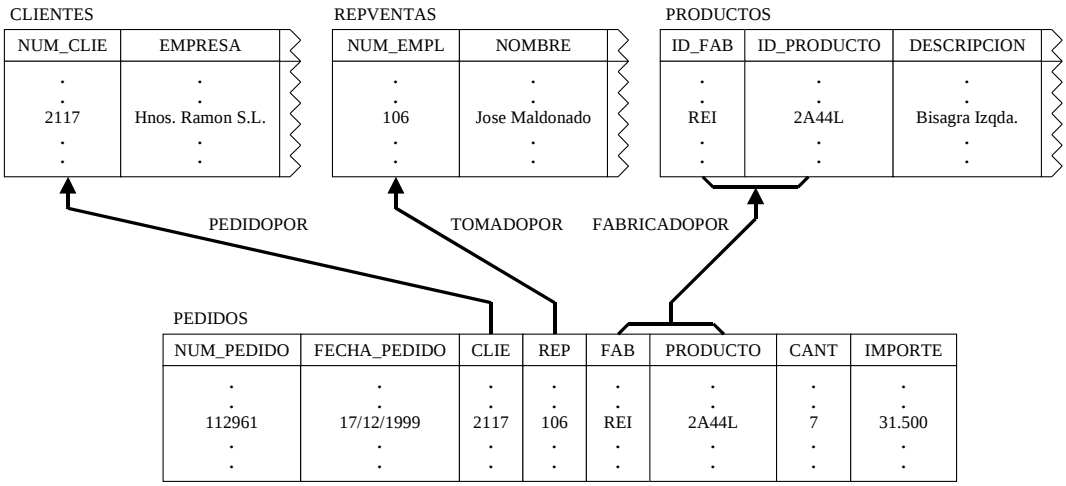

Figura 5.3.3.1: Relaciones y nombres de relaciones creadas por la definición anterior.

Cuando el gestor de la base de datos procesa la sentencia CREATE TABLE, compara cada definición de clave foránea con la definición de la tabla que referencia, asegurándose que la clave foránea y la clave primaria de la tabla referenciada concuerden en el número de columnas que contienen y en sus tipos de datos. La tabla referenciada debe estar ya definida en la base de datos para que esta comparación tenga éxito.

Si dos o más tablas forman un ciclo referencial (ambas se referencian mutuamente), no se puede definir la clave foránea de la tabla que se cree en primer lugar, ya que la tabla referenciada todavía no existe. En su lugar debe crearse la tabla sin definición de clave foránea y añadir la clave foránea posteriormente utilizando la sentencia ALTER TABLE.

5.3.4 Restricciones de unicidad.

El estándar ANSI/ISO específica que las restricciones de unicidad, o sea, el hecho de que un valor sin ser clave deba ser único, también se definen en la sentencia CREATE TABLE, utilizando para ello la cláusula UNIQUE. Por ejemplo, la siguiente sentencia CREATE TABLE exige que los valores de ciudad sean únicos:

```
CREATE TABLE oficinas
        (oficina INTEGER NOT NULL,
        ciudad VARCHAR(15) NOT NULL,
        region VARCHAR(10) NOT NULL,
        objetivo INTEGER,
        ventas INTEGER NOT NULL,
        PRIMARY KEY (oficina),
        UNIQUE (ciudad))
```

OFICINA	CIUDAD	REGION	OBJETIVO	VENTAS

Tabla 5.3.4.1: Definición de la tabla OFICINAS con una restricción de unicidad.

5.4 Eliminación de una tabla.

Una base de datos, en su intento de representar la realidad, sufrirá modificaciones a lo largo de su existencia, se añadirán nuevas tablas y otras tablas ya no

serán necesarias. Se puede elimina una tabla de la base de datos mediante la sentencia DROP TABLE, cuyo diagrama sintáctico se especifica en la figura siguiente:

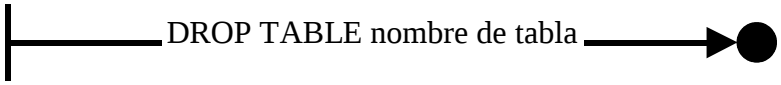

Figura 5.4.1: Diagrama sintáctico de la sentencia DROP TABLE.

El nombre de la tabla en la sentencia indica la tabla a eliminar. Normalmente se eliminará una de las tablas propias del usuario y se utilizará el nombre de tabla no cualificado. Con el permiso adecuado, también se pueden eliminar tablas propiedad de otro usuario especificando un nombre de tabla cualificado. Algunos ejemplos de la sentencia DROP TABLE son:

```
DROP TABLE clientes
```

Tabla 5.4.1: Supresión de la tabla clientes de la base de datos.

```
DROP TABLE quique.cumpleaños
```

Tabla 5.4.2: Supresión de la tabla cumpleaños del usuario quique (siempre y cuando se tenga permiso).

Cuando se ejecuta una sentencia DROP TABLE todos los contenidos se pierden y no existe forma de recuperar los datos.

5.5 Modificación de la definición de una tabla.

Después de utilizar un tiempo una tabla, los usuarios suelen descubrir que desean almacenar información adicional con respecto a las entidades representadas en la tabla. Por ejemplo pueden desear:

- Añadir el nombre y número de teléfono de una persona de contacto a cada fila de la tabla CLIENTES.

- Añadir una columna de nivel de inventario mínimo a la tabla PRODUCTOS, para que la base de datos pueda alertar automáticamente cuando el stock de un producto esté bajo.

- Eliminar la información relativa al objetivo de ventas en la tabla OFICINAS, al ser esté la suma de los objetivos de los representantes de la oficina.

Todos estos cambios pueden realizarse con la sentencia ALTER TABLE, cuyo diagrama sintáctico se muestra en la figura 5.5.1. Al igual que la sentencia DROP TABLE, ALTER TABLE actuará normalmente sobre tablas propias, aunque, con el permiso adecuado, es posible alterar tablas de otros usuarios.

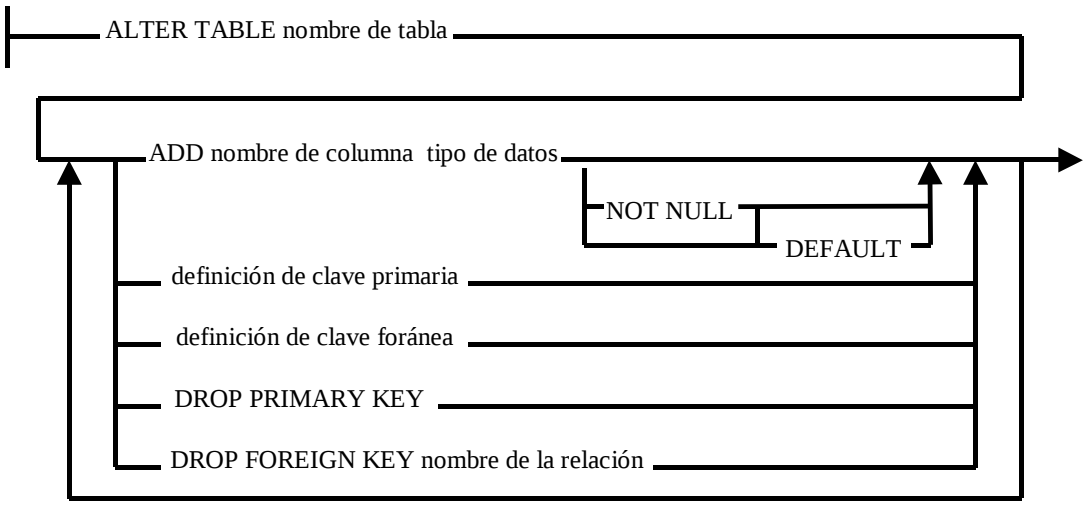

Figura 5.5.1: Diagrama sintáctico de la sentencia ALTER TABLE.

5.5.1 Adición de una columna.

El uso más común de la sentencia ALTER TABLE es añadir una columna a una tabla existente. La cláusula de definición de la columna en la sentencia ALTER TABLE es virtualmente idéntica a la de la sentencia CREATE TABLE y funciona del mismo modo. La nueva columna se añade al final de las definiciones de columna de la tabla y aparece como la columna más a la derecha en consultas posteriores.

Por defecto, el gestor de la base de datos supone un valor NULL para la columna recién añadida en todas las filas existentes en la tabla. Si la columna se declara NOT NULL DEFAULT, el gestor supone que el valor por omisión es el del tipo de datos de la columna. Obsérvese que no es posible declarar simplemente la columna como NOT NULL, pues en tal caso el gestor de la base de datos asumiría por defecto valores NULL a la columna, violando la restricción impuesta.

Algunos ejemplos de sentencias ALTER TABLE son:

```
ALTER TABLE clientes ADD nombre_contacto VARCHAR(30)
```

NUM_CLIE	EMPRESA	REP_CLIE	LIMITE_CREDITO	NOMBRE_CONTACTO

Tabla 5.5.1.1: Añade un nombre de contacto a la tabla CLIENTES.

```
ALTER TABLE clientes ADD telefono_contacto CHAR(9)
```

NUM_CLIE	EMPRESA	REP_CLIE	LIMITE_CREDITO	TELEFONO_CONTACTO

Tabla 5.5.1.2: Añade un teléfono de contacto a la tabla CLIENTES.

```
ALTER TABLE productos ADD cant_min INTEGER NOT NULL DEFAULT 0
```

ID_FAB	ID_PRODUCTO	DESCRIPCION	PRECIO	EXISTENCIAS	CANT_MIN

Tabla 5.5.1.3: Añade una columna inventario mínimo a la tabla PRODUCTOS.

5.5.2 Supresión de una columna.

La sentencia ALTER TABLE no se puede utilizar para eliminar una columna existente, y de hecho SQL no proporciona un método de eliminación de una columna. La forma de eliminar una columna, si realmente se desea es el siguiente procedimiento:

1. Crear una nueva tabla temporal mediante CREATE TABLE sin la columna no deseada.

2. Copiar todos los datos menos los de esa columna mediante un INSERT multifila.

3. Borrar la definición de la tabla anterior mediante la sentencia DROP TABLE.

4. Crear nuevamente la tabla sin la columna no deseada mediante CREATE TABLE.

5. Volver a insertar los datos en la tabla desde la tabla temporal mediante un INSERT multifila.

6. Eliminar la tabla temporal mediante DROP TABLE.

5.5.3 Modificación de claves primaria y foránea.

Otro uso habitual de ALTER TABLE es cambiar o añadir definiciones de clave primaria y clave foránea a una tabla. Las sentencias que añaden definiciones de claves primarias o foráneas a una tabla son exactamente iguales que las de la sentencia CREATE TABLE y funcionan del mismo modo.

Las cláusulas que crean o suprimen una clave primaria o una clave foránea son sencillas, tal y como puede verse en los siguientes ejemplos:

```
ALTER TABLE oficinas FOREIGN KEY enregion(region) REFERENCES regiones
```

Sentencia 5.5.3.1: Hace la columna región de la tabla OFICINAS una clave foránea de una nueva tabla llamada REGIONES que contiene las regiones existentes.

```
ALTER TABLE repventas DROP FOREIGN KEY operaen
ALTER TABLE oficinas DROP PRIMARY KEY
```

Sentencia 5.5.3.2: Elimina la clave primaria de la tabla OFICINAS.

En el segundo ejemplo se puede observar como para poder eliminar la clave primaria de la tabla OFICINAS hemos de eliminar la clave foránea de la tabla REPVENTAS. Además, es conveniente resaltar que la clave foránea solo puede ser eliminada si la relación tiene nombre, en caso contrario la tabla deberá ser tratada como en el caso anterior de supresión de una columna.

5.6 Vistas en SQL.

En SQL, una vista es una tabla virtual en la base de datos cuyos contenidos están definidos por una consulta. Para el usuario de la base de datos, la vista aparece igual que

una tabla real, con un conjunto de columnas designadas y filas de datos. Pero a diferencia de una tabla real, una vista no existe en la base de datos como conjunto almacenado de valores. En su lugar, las filas y columnas de datos visibles a través de la vista son los resultados producidos por la consulta que define la vista.

Las vistas proporcionan una variedad de beneficios, pudiendo resaltarse los siguientes:

- Seguridad. Cada usuario puede obtener permiso para acceder a la base de datos únicamente a través de un pequeño conjunto de vistas que contienen los datos específicos que el usuario está autorizado a ver.

- Simplicidad de consulta. Una vista puede extraer datos de varias tablas diferentes y presentarlos como una única tabla, haciendo que consultas multitabla se formulen como consultas de una sola tabla con respecto a la vista.

- Simplicidad estructurada. Las vistas pueden dar a un usuario una visión "personalizada" de la estructura que tiene la base de datos presentando está como un conjunto de tablas virtuales que tienen sentido para ese usuario.

- Aislamiento frente al cambio. Una vista representa una imagen consistente inalterada de la base de datos, incluso si las tablas fuente subyacentes se dividen, reestructuran o cambian de nombre.

- Integridad de datos. Si los datos se acceden y se introducen a través de una vista, el gestor de la base de datos puede comprobar automáticamente los datos para asegurarse que satisfacen restricciones de integridad específicas.

Sin embargo, las vistas presentan también una serie de desventajas al utilizarlas en lugar de una tabla real. Estas desventajas son:

- Rendimiento. Las vistas crean la apariencia de una tabla, pero el gestor de la base de datos debe traducir las consultas con respecto a la vista en consultas con respecto a las tablas fuente subyacentes.

- Restricciones de actualización. Cuando un usuario trata de actualizar filas de una vista, el gestor de la base de datos debe traducir la petición a una actualización sobre las filas de las tablas fuente. Esto es posible para vistas sencillas, pero vistas complejas no pueden ser actualizadas, son "de solo lectura".

5.6.1 Creación de una vista en SQL.

La sentencia CREATE VIEW, cuyo diagrama sintáctico se muestra a continuación, se utiliza para crear una vista. La sentencia asigna un nombre a la vista y especifica la consulta que define la vista. Para crear la vista con éxito, es preciso tener permiso para acceder a todas las tablas referenciadas en la consulta.

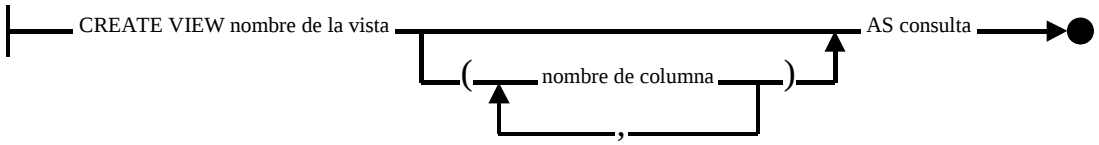

Figura 5.6.1.1: Diagrama sintáctico de la sentencia CREATE VIEW.

La sentencia CREATE VIEW puede asignar opcionalmente un nombre a cada columna en la vista recién creada. Si se especifica una lista de nombres de columnas, debe tener el mismo número de elementos que el número de columnas producido por la consulta.

Aunque las vistas se crean del mismo modo, en la práctica diferentes tipos de vistas son utilizados para propósitos diferentes. Los próximos puntos examinan estos tipos de vistas y proporcionan ejemplos de la sentencia CREATE VIEW.

5.6.2 Vistas horizontales.

Un uso de las vistas es restringir el acceso de un usuario a un conjunto de filas seleccionadas de una tabla. Por ejemplo, podríamos querer que un director de ventas vea solamente las filas de la tabla REPVENTAS correspondientes a los vendedores de la región del director. Para lograr esto, se pueden definir dos vistas, de la siguiente forma, que permiten a cada director ver tan solo los vendedores de su región:

```
CREATE VIEW repeste AS SELECT * FROM repventas WHERE oficina_rep IN (11,
12, 13)
```

NUM_EMPL	NOMBRE	EDAD	OFICINA_REP	TITULO	CONTRATO
106	Jose Maldonado	52	11	VP Ventas	14/06/1998
104	Carlos Martinez	33	12	Dir. Ventas	19/05/1997
105	Belen Aguirre	37	13	Dir. Ventas	12/02/1998
109	Maria Garcia	31	11	Rep. Ventas	12/10/1999
101	Daniel Gutierrez	45	12	Rep. Ventas	20/10/1996
103	Pedro Cruz	29	12	Rep. Ventas	01/03/1997

DIRECTOR	CUOTA	VENTAS
NULL	25.000	32.958
106	17.500	0
104	30.000	39.327
106	27.500	7.105
104	27.500	26.628
104	25.000	2.700

Tabla 5.6.2.1: Creación de una vista con los vendedores de la zona Este.

```
CREATE VIEW repcentro AS SELECT * FROM repventas WHERE oficina_rep IN (21,
22)
```

NUM_EMPL	NOMBRE	EDAD	OFICINA_REP	TITULO	CONTRATO
108	Lorenzo Fernandez	62	21	Dir. Ventas	12/10/1999
102	Soledad Martinez	48	21	Rep. Ventas	10/12/1996
107	Natalia Martin	49	22	Rep. Ventas	14/11/1998

DIRECTOR	CUOTA	VENTAS
106	30.000	58.533
108	30.000	22.776

108	27.500	34.432

Tabla 5.6.2.2: Creación de una vista con los vendedores de la zona Centro.

Ahora se puede dar a cada director de ventas permiso para acceder a la vista adecuada, negando el acceso a la otra vista.

Una vista como REPESTE o REPCENTRO se denomina vista horizontal pues como hemos visto, divide horizontalmente la tabla fuente para crear la vista. Todas las columnas de la tabla fuente participan en la vista, pero sólo algunas de sus filas con visibles a través de la vista.

Otros ejemplos de vistas horizontales son:

```
CREATE VIEW oficinaeste AS SELECT * FROM oficinas WHERE region='Este'
```

OFICINA	CIUDAD	REGION	DIR	OBJETIVO	VENTAS
11	Valencia	Este	106	52.500	40.063
12	Barcelona	Este	104	70.000	29.328
13	Alicante	Este	105	30.000	39.327

Tabla 5.6.2.3: Creación de una vista que contiene únicamente las oficinas de la región Este.

```
CREATE VIEW pedidossoledad AS SELECT * FROM pedidos WHERE clie IN
(SELECT num_clie FROM clientes WHERE rep_clie=102)
```

NUM_PEDIDO	FECHA_PEDIDO	CLIE	REP	FAB	PRODUCTO	CANT	IMPORTE
113024	20/01/2000	2114	108	QSA	XK47	20	7.100
112979	12/10/1999	2114	102	ACI	4100Z	6	15.000
113048	10/02/2000	2120	102	IMM	779C	2	3.750
112993	04/01/2000	2106	102	REI	2A45C	24	1.896
113065	27/02/2000	2106	102	QSA	XK47	6	2.130

Tabla 5.6.2.4: Creación de una vista para Soledad Martinez (empleada 102) que contiene solamente los pedidos remitidos por clientes asignados a ella.

```
CREATE VIEW clientesvip AS SELECT * FROM clientes WHERE 30000<(SELECT
SUM(importe) FROM pedidos WHERE clie=num_clie)
```

NUM_CLIE	EMPRESA	REP_CLIE	LIMITE_CREDITO
2117	Hnos. Ramon S.L.	106	35.000
2112	Lopez Asociados S.L.	108	50.000
2103	Pino S.L.	105	50.000
2109	Roda & Castedo S.L.	103	25.000

Tabla 3.19.2.5: Creación de una vista que muestra aquellos clientes que tiene más de 30.000 en pedidos.

5.6.3 Vistas verticales.

Otro uso habitual de las vistas es restringir el acceso de un usuario a sólo ciertas columnas de una tabla. Por ejemplo, puede desearse que el departamento de procesamiento de pedidos solo pueda acceder al número de empleado, su nombre y la asignación de oficina de cada vendedor, pues es información necesaria para procesar correctamente el pedido, pero no hay necesidad de que acceda al total de ventas

realizadas o a la cuota. Esta vista selectiva de la tabla REPVENTAS puede construirse con la vista siguiente:

```
CREATE VIEW inforep AS SELECT num_empl, nombre, oficina_rep FROM repventas
```

NUM_EMPL	NOMBRE	OFICINA_REP
106	Jose Maldonado	11
104	Carlos Martinez	12
105	Belen Aguirre	13
109	Maria Garcia	11
108	Lorenzo Fernandez	21
102	Soledad Martinez	21
101	Daniel Gutierrez	12
110	Antonio Valle	NULL
103	Pedro Cruz	12
107	Natalia Martin	22

Tabla 5.6.3.1: Creación de una vista con información seleccionada de cada vendedor.

Una vista como la anterior se denomina vista vertical, pues divide la tabla fuente verticalmente para crear la vista. Otros ejemplos de vistas verticales son:

```
CREATE VIEW infooficina AS SELECT oficina,ciudad,region FROM oficinas
```

OFICINA	CIUDAD	REGION
22	Toledo	Centro
11	Valencia	Este
12	Barcelona	Este
13	Alicante	Este
21	Madrid	Centro

Tabla 5.6.3.2: Creación de una vista vertical de la tabla OFICINAS.

```
CREATE VIEW infoclie AS SELECT empresa,rep_clie FROM clientes
```

EMPRESA	REP_CLIE
EVBE S.A.	103
Exclusivas del Este S.L.	101
Pino S.L.	105
Hermanos Martinez S.A.	102
Distribuciones Sur S.A.	110
AFS S.A.	101
Exclusivas Soriano S.A.	106
Lopez Asociados S.L.	108
Hernandez & hijos S.L.	103
Componentes Fernandez S.A.	102
Domingo S.L.	107
Zapater Importaciones S.A.	109
Hnos. Ramon S.L.	106
JPF S.L.	105
Distribuciones Montiel S.L.	102
Construcciones Leon S.A.	102
Martinez & Garcia S.L.	109
Exclusivas Norte S.A.	108
Ian & Shmidt	104
Roda & Castedo S.L.	103
MALB S.A.	101

Tabla 5.6.3.3: Creación de una vista vertical de la tabla CLIENTES.

5.6.4 Vistas con subconjuntos fila/columna.

Cuando se define una vista, SQL no restringe a divisiones puramente horizontales o verticales de una tabla. De hecho, SQL no posee la noción de vistas horizontales y verticales, sino que estos conceptos meramente ayudan a visualizar cómo la vista presenta la información a partir de la tabla fuente. Por ello, es posible definir una vista que divida la tabla fuente tanto por la dimensión horizontal como por la vertical, como en el ejemplo siguiente:

```
CREATE VIEW cliebelen AS SELECT num_clie,empresa,limite_credito FROM clientes
WHERE rep_clie=105
```

NUM_CLIE	EMPRESA	LIMITE_CREDITO
2103	Pino S.L.	50.000
2122	JPF S.L.	30.000

Tabla 5.6.4.1: Creación de una vista que contiene el número de cliente, el nombre de la empresa y el limite de crédito para los clientes de Belen Aguirre (empleado número 105).

5.6.5 Vistas agrupadas.

La consulta especificada en una definición de una vista puede incluir una cláusula GROUP BY. Este tipo de vista se denomina vista agrupada, ya que los datos visibles a través de ella son el resultado de una consulta agrupada. Las vistas agrupadas efectúan la misma función que las consultas agrupadas; agrupan filas relacionadas de datos y producen una fila de resultados de consulta para cada grupo, sumarizando los datos de ese grupo. Veamos un ejemplo:

```
CREATE VIEW ped_por_rep(quien,cuantos,total,inf,sup,medio) AS SELECT
rep,COUNT(*),SUM(importe),MIN(importe),MAX(importe),AVG(importe) FROM
pedidos GROUP BY rep
```

QUIEN	CUANTOS	TOTAL	INF	SUP	MEDIO
106	2	32.958	1.458	31.500	16.479,00
105	5	39.327	702	27.500	7.865,40
108	7	58.633	652	45.000	8.376,14
101	3	26.628	150	22.500	8.876,00
110	2	23.132	632	22.500	11.566,00
109	2	7.105	1.480	5.625	3.552,50
107	3	34.432	652	31.350	11.477,33
102	4	22.776	1.896	15.000	5.694,00
103	2	2.700	600	2.100	1.350,00

Tabla 5.6.5.1: Creación de una vista que contiene datos sumarios de los pedidos para cada vendedor.

Una vez que se define una vista agrupada, puede ser utilizada para simplificar consultas. Por ejemplo, la siguiente consulta genera un informe que sumariza los pedidos de cada vendedor:

```
SELECT nombre,cuantos,total,medio FROM repventas,ped_por_rep WHERE
quien=num_empl ORDER BY total DESC
```

NOMBRE	CUANTOS	TOTAL	MEDIO

Lorenzo Fernandez	7	58.633	8.376,14
Belen Aguirre	5	39.327	7.865,40
Natalia Martin	3	34.432	11.477,33
Jose Maldonado	2	32.958	16.479,00
Daniel Gutierrez	3	26.628	8.876,00
Antonio Valle	2	23.132	11.566,00
Soledad Martinez	4	22.776	5.694,00
Maria Garcia	2	7.105	3.552,50
Pedro Cruz	2	2.700	1.350,00

Tabla 5.6.5.2: Datos sumarios de los vendedores obtenidos a partir de la vista PED_POR_REP.

5.6.6 Vistas compuestas.

Una de las razones más frecuentes para utilizar vistas es simplificar las consultas multitabla. Especificando una consulta de dos o tres tablas en la definición de la vista, se puede crear una vista compuesta que extrae sus datos de dos o tres tablas diferentes y presenta los resultados de la consulta como una única tabla virtual. Por ejemplo, supongamos que se efectúan con frecuencia consultas sobre la tabla PEDIDOS de la base de datos ejemplo y se desea trabajar con los nombres de los empleados y los clientes en vez de con sus códigos. La siguiente vista satisface esa demanda:

```
CREATE VIEW info_pedido(num_pedido,empresa,nombre_rep,importe) AS SELECT
num_pedido,empresa,nombre,importe FROM pedidos,clientes,repventas WHERE
clie=num_clie AND rep=num_empl
```

NUM_PEDIDO	EMPRESA	NOMBRE_REP	IMPORTE
112961	Hnos. Ramon S.L.	Jose Maldonado	31.500
113012	EVBE S.A.	Belen Aguirre	3.745
112989	Exclusivas Soriano S.A.	Jose Maldonado	1.458
113051	Exclusivas Norte S.A.	Lorenzo Fernandez	1.420
112968	Exclusivas del Este S.L.	Daniel Gutierrez	3.978
110036	Distribuciones Sur S.A.	Antonio Valle	22.500
113045	Lopez Asociados S.L.	Lorenzo Fernandez	45.000
112963	Pino S.L.	Belen Aguirre	3.276
113013	Exclusivas Norte S.A.	Lorenzo Fernandez	652
113058	Zapater Importaciones S.A.	Maria Garcia	1.480
112997	Domingo S.L.	Natalia Martin	652
112983	Pino S.L.	Belen Aguirre	702
113024	Componentes Fernandez S.A.	Lorenzo Fernandez	7.100
113062	Domingo S.L.	Natalia Martin	2.430
112979	Componentes Fernandez S.A.	Soledad Martinez	15.000
113027	Pino S.L.	Belen Aguirre	4.104
113007	Lopez Asociados S.L.	Lorenzo Fernandez	2.825
113069	Roda & Castedo S.L.	Natalia Martin	31.350
113034	Distribuciones Sur S.A.	Antonio Valle	632
112992	Exclusivas Norte S.A.	Lorenzo Fernandez	760
112975	EVBE S.A.	Pedro Cruz	2.100
113055	Zapater Importaciones S.A.	Daniel Gutierrez	150
113048	Distribuciones Montiel S.L.	Soledad Martinez	3.750
112993	Construcciones Leon S.A.	Soledad Martinez	1.896
113065	Construcciones Leon S.A.	Soledad Martinez	2.130
113003	Zapater Importaciones S.A.	Maria Garcia	5.625
113049	Exclusivas Norte S.A.	Lorenzo Fernandez	776
112987	Pino S.L.	Belen Aguirre	27.500
113057	EVBE S.A.	Pedro Cruz	600
113042	Importaciones Martin S.L.	Daniel Gutierrez	22.500

Tabla 5.6.6.1: Creación de una vista de la tabla PEDIDOS con nombres en vez de códigos.

Entonces, podemos recurrir a esta vista para generar informes de forma más sencilla. Por ejemplo:

```
SELECT empresa,importe,nombre_rep FROM info_pedido WHERE importe>20000
ORDER BY importe DESC
```

EMPRESA	IMPORTE	NOMBRE_REP
Lopez Asociados S.L.	45.000	Lorenzo Fernandez
Hnos. Ramon S.L.	31.500	Jose Maldonado
Roda & Castedo S.L.	31.350	Natalia Martin
Pino S.L.	27.500	Belen Aguirre
Distribuciones Sur S.A.	22.500	Antonio Valle
Importaciones Martin S.L.	22.500	Daniel Gutierrez

Tabla 5.6.6.2: Mayores pedidos realizados, ordenados por importe.

La vista hace mucho más fácil de examinar lo que sucede en la consulta que si fuera expresada como la composición equivalente de tres tablas.

5.7 Seguridad en SQL.

La implementación de un esquema de seguridad y el reforzamiento de las restricciones de seguridad son responsabilidad del software gestor de la base de datos. El lenguaje SQL supone un nivel general de seguridad del software gestor de la base de datos y sus sentencias se utilizan para especificar restricciones de seguridad. El esquema de seguridad SQL se basa en tres conceptos:

- Los usuarios son los actores de la base de datos. Cada vez que el gestor de la base de datos recupera, inserta, suprime o actualiza datos, lo hace a cuenta de algún usuario. El gestor de la base de datos permitirá o prohibirá la acción dependiendo de qué usuario esté efectuando la petición.

- Los objetos de la base de datos son los elementos a los cuales se puede aplicar la protección de seguridad SQL. La seguridad se aplica generalmente a tablas y vistas, pero otros objetos tales como formularios, programas de aplicación y bases de datos enteras también puede ser protegidos. La mayoría de usuarios tendrán permiso para utilizar ciertos objetos de la base de datos y tendrán prohibido el uso de otros.

- Los privilegios son las acciones que un usuario tiene permitido efectuar para un determinado objeto de la base de datos. Un usuario puede tener permiso para SELECT e INSERT sobre filas en una tabla determinada, pero puede carecer de permiso para DELETE o UPDATE filas de la tabla. Cada usuario diferente puede tener un conjunto diferente de privilegios.

La creación y eliminación de usuarios en SQL no es estándar, dependiendo de cada producto comercial, por lo cual no se comentara aquí. Sin embargo, la concesión y revocación de privilegios si es estándar y está recogida en las sentencias GRANT y REVOKE.

5.7.1 Concesión de privilegios en SQL. La sentencia GRANT.

La sentencia GRANT, cuyo diagrama sintáctico se muestra a continuación, se utiliza para conceder privilegios de seguridad sobre objetos de la base de datos a usuarios específicos. Normalmente la sentencia GRANT es utilizada por el propietario de la tabla o vista para proporcionar a otros usuarios acceso a los datos.

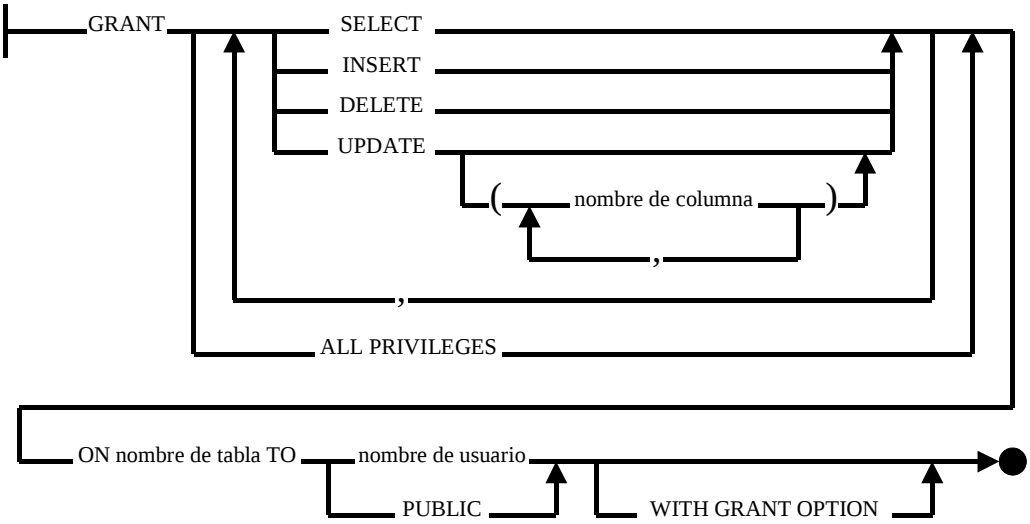

Figura 5.7.1.1: Diagrama sintáctico de la sentencia GRANT.

Como puede verse, la sentencia GRANT incluye una lista específica de los privilegios a conceder, el nombre de la tabla a la cual se aplican los privilegios y el usuario al cual se conceden los privilegios. Algunos ejemplos son:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON pedidos TO usuarioA
```

Sentencia 5.7.1.1: Concede al usuario A acceso completo a la tabla PEDIDOS.

```
GRANT SELECT, INSERT ON clientes TO usuarioA
GRANT SELECT ON clientes TO usuarioB
```

Sentencia 5.7.1.2: Concede al usuario A permisos de selección e inserción y al usuario B permisos de selección en la tabla CLIENTES.

```
GRANT ALL PRIVILEGES ON repventas TO usuarioA
```

Sentencia 5.7.1.3: Concede todos los privilegios sobre la tabla REPVENTAS al usuario A.

```
GRANT SELECT ON oficinas TO PUBLIC
```

Sentencia 5.7.1.4: Concede privilegios de selección a todos los usuarios de la base de datos en la tabla OFICINAS.

```
GRANT SELECT ON oficinas TO usuarioA WITH GRANT OPTION
```

Sentencia 5.7.1.5: Concede privilegios de selección al usuario A, permitiendo además que dicho usuario pueda transmitir dicho privilegio a otros usuarios.

5.7.2 Revocación de privilegios en SQL. La sentencia REVOKE.

En la mayoría de bases de datos basadas en SQL, los privilegios que se han concedido con la sentencia GRANT pueden ser retirados con la sentencia REVOKE, cuyo diagrama sintáctico se muestra a continuación.

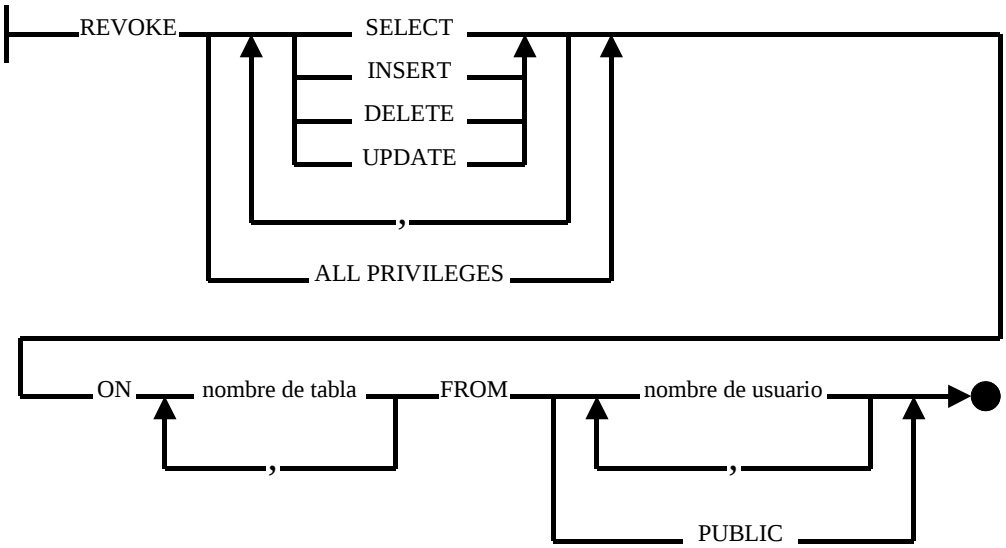

Figura 5.7.2.1: Diagrama sintáctico de la sentencia REVOKE.

La sentencia REVOKE tiene una estructura que se asemeja estrechamente a la sentencia GRANT, especificando un conjunto específico de privilegios a ser revocados, para un objeto de la base de datos específico, para uno o más usuarios. Una sentencia REVOKE puede retirar todos o parte de los privilegios que previamente se han concedido a un usuario. Es necesario especificar que un usuario solo puede retirar los privilegios que él mismo ha concedido a otro usuario. Algunos ejemplos son:

```
REVOKE SELECT, UPDATE ON pedidos FROM usuarioA
```

Sentencia 5.7.2.1: Elimina los permisos de selección y actualización de la tabla PEDIDOS al usuario A.

```
REVOKE ALL PRIVILEGES ON oficinas FROM usuarioA
```

Sentencia 5.7.2.2: Elimina todos los privilegios concedidos sobre la tabla OFICINAS al usuario A.

```
REVOKE ALL PRIVILEGES ON oficinas FROM PUBLIC
```

Sentencia 5.7.2.3: Elimina todos los privilegios sobre la tabla OFICINAS a todos los usuarios.