



Guía de Laboratorio N°03

“Introducción a SQLITE”

Objetivos:

Que el estudiante aprenda los conceptos básicos del lenguaje SQLite, el uso de TRIGGERS(para definición de restricciones de integridad o para modificar valores en otras tablas), creación de base de datos y tablas que la conforman, así como las operaciones básicas de consulta a base de datos INSERT, DELETE, UPDATE y SELECT.

Descripción:

En esta práctica se creará una base de datos usando lenguaje SQLite, se definirán las tablas y sus propiedades, se realizaran operaciones básicas para insertar, borrar, modificar y actualizar datos sobre ella utilizando la herramienta SQLite Administrator.



Índice

¿Qué es SQLite?	1
Aplicación.....	2
CREAR TABLAS	4
INSERTAR REGISTROS.....	11
MODIFICAR REGISTROS	14
CONSULTAR REGISTROS.....	15
ELIMINAR REGISTROS.....	16
TRIGGERS	17
RESTRICCION DE INTEGRIDAD REFERENCIAL(Foreign Key).....	23



¿Qué es SQLite?

SQLite es una librería escrita en lenguaje C que implementa un manejador de base de datos SQL embebido.

SQLite es un motor de bases de datos muy popular en la actualidad por ofrecer características tan interesantes como su pequeño tamaño, no necesitar servidor, precisar poca configuración, ser transaccional y además es de código libre.

SQLite soporta las características estándar de bases de datos relacionales, como la sintaxis SQL, transacciones y declaraciones preparadas. Además, sólo requiere un poco de memoria en tiempo de ejecución aproximadamente 250 Kb.

A diferencia de los sistemas de gestión de bases de datos cliente-servidor, el motor de SQLite no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, la biblioteca SQLite se enlaza con el programa pasando a ser parte integral del mismo. El programa utiliza la funcionalidad de SQLite a través de llamadas simples a subrutinas y funciones. Esto reduce la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados como un sólo archivo estándar en la máquina host. Este diseño simple se logra bloqueando todo el archivo de base de datos al principio de cada transacción.

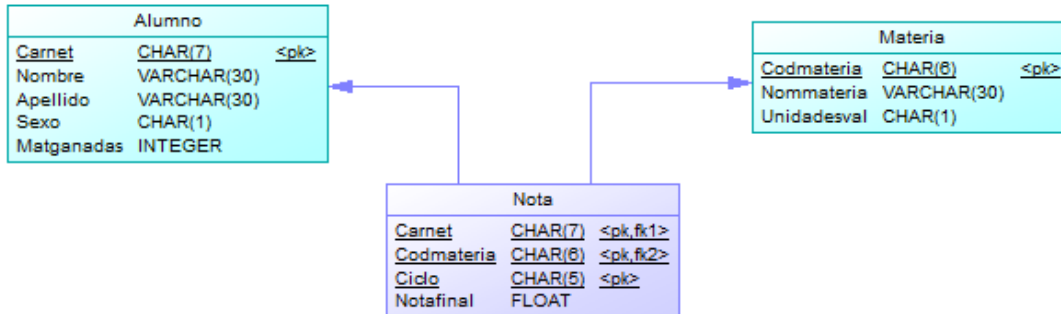
Los programas que se enlacen con la librería SQLite pueden tener acceso a una base de datos SQL, sin tener que ejecutar un programa de RDBMS separado.

SQLite soporta el tipo de datos TEXT similar a String en Java, INTEGER similar a long en Java y REAL similar a double en Java. Todos los demás tipos se deben convertir en uno de estos campos antes de guardarlos en la base de datos. SQLite no valida si los tipos de escritos a las columnas son en realidad del tipo definido, por ejemplo usted puede escribir un número entero en una columna de cadena y viceversa.



Aplicación

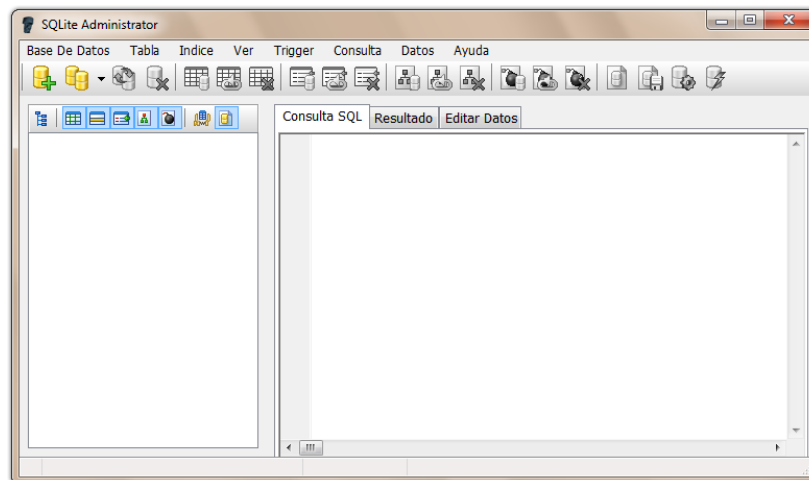
Para crear una base de datos a partir del siguiente *modelo fisico*.




BASE DE DATOS ALUMNO. MODELO FISICO

Nota: Para realizar la práctica de una manera más profesional (y que esto le sirva para su proyecto)
1) Puede consultar la guía1 de bases de datos 2012 (en esta aula virtual) y llevar el modelo desde conceptual (ver anexo) hasta fisico.

Ejecuta [sqliteadmin.exe](#), veras una ventana como esta:



Empezaremos creando una base de datos nueva. A diferencia de Oracle o MySQL, SQLite permite crear bases de dato de una manera fácil. Para ello presiona el icono de **Nuevo**  o bien en el menú **Base De Datos -> Nuevo**.

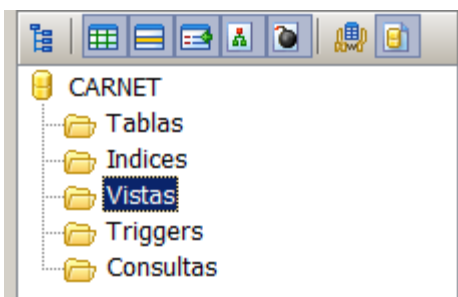
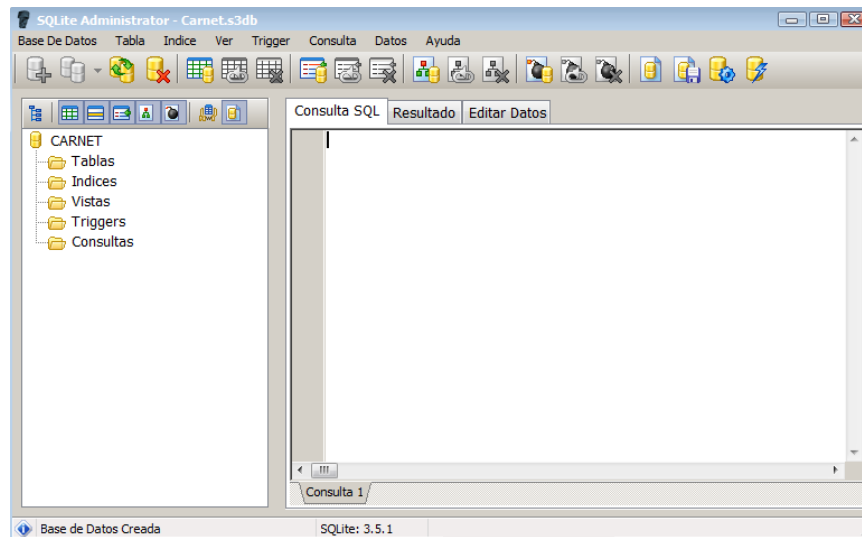


UNIVERSIDAD DE EL SALVADOR
FACULTAD DE INGENIERIA Y ARQUITECTURA
ESCUELA DE INGENIERIA DE SISTEMAS INFORMATICOS
PROGRAMACION PARA DISPOSITIVOS MOVILES
PDM115

Ciclo I-2019

Escribe el nombre **Carnet (su carnet)** y selecciona la ubicación de destino en la que será guardado, preferentemente en la opción TIPO selecciona la versión **SQLite3 DB** y para finalizar presiona el botón **Guardar**. Nota que SQLite Administrator nombra el archivo de la base de datos automáticamente con la extensión ***.s3db** para saber que es una base de datos versión SQLite 3.

Ahora la ventana tendrá un aspecto como este:



Los botones de la parte lateral izquierda permiten asignar la visibilidad de los diferentes elementos de la base de datos, ya que no hemos agregado nada aun, todas las carpetas estarán vacías inicialmente.

Nos centraremos en la carpeta Tablas, que es, como su nombre lo indica donde podremos observar todas las tablas que formen parte de la base de datos.

Los botones de la parte superior nos permiten realizar todas las acciones referentes a la administración de la base de datos actual, como ejecutar una consulta SQL, crear, editar o eliminar una tabla desde un asistente, etc.





Las tres pestaña ubicadas debajo de los controles mencionados antes, permiten navegar entre las acciones que podemos realizar, como es realizar una **Consulta SQL** a la base de datos, ver los **Resultados** que devuelve la última consulta ejecutada, o bien **Editar Datos** de las tablas, ya sea para ingresar, modificar o eliminar un registro directamente mediante la interfaz. Nuevamente las acciones de insertar, modificar o eliminar las realizaremos a partir de comandos SQL y no desde la interfaz que nos proporciona SQLite Administrator en la pestaña **Editar Datos**.

CREAR TABLAS

Para las tablas, usaremos la forma más simple, sin embargo es posible definir muchas más propiedades a la hora de ejecutar el código para crear una tabla (puedes ver más acerca de esto en la documentación oficial de SQLite <http://www.sqlite.org>) Las llaves foráneas son posibles de ser implementadas en SQLite, pero no las trabaja correctamente, por lo que las definimos a la hora de crear las tablas, sin embargo debemos respetar siempre el concepto de Llaves primarias y Llaves foráneas.

Para la base de datos Alumno se tiene lo siguiente:

```
CREATE TABLE alumno (  
    carnet VARCHAR(7) NOT NULL PRIMARY KEY,  
    nombre VARCHAR(30),  
    apellido VARCHAR(30),  
    sexo VARCHAR(1),  
    matganadas INTEGER);  
CREATE TABLE materia (  
    codmateria VARCHAR(6) NOT NULL PRIMARY KEY,  
    nommateria VARCHAR(30),  
    unidadesval VARCHAR(1));  
CREATE TABLE nota (  
    carnet VARCHAR(7) NOT NULL ,  
    codmateria VARCHAR(6) NOT NULL ,  
    ciclo VARCHAR(5) ,  
    notafinal FLOAT ,  
    PRIMARY KEY(carnet,codmateria,ciclo)  
    CONSTRAINT fk_nota_materia FOREIGN KEY (codmateria) REFERENCES  
materia(codmateria) ON DELETE RESTRICT,  
    CONSTRAINT fk_nota_alumno FOREIGN KEY (carnet) REFERENCES alumno(carnet) ON  
DELETE RESTRICT)
```




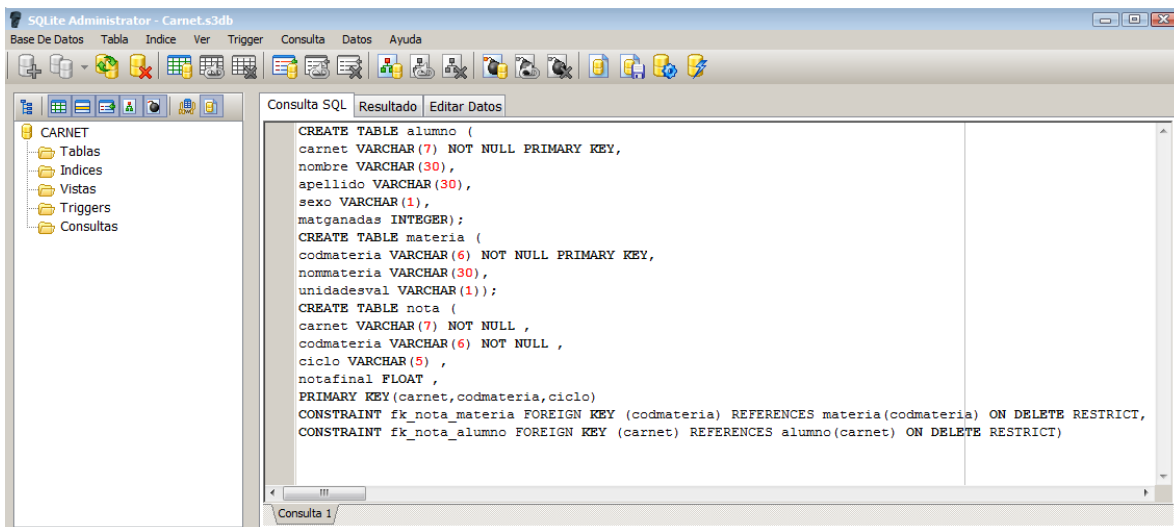
UNIVERSIDAD DE EL SALVADOR
FACULTAD DE INGENIERIA Y ARQUITECTURA
ESCUELA DE INGENIERIA DE SISTEMAS INFORMATICOS
PROGRAMACION PARA DISPOSITIVOS MOVILES
PDM115

Ciclo I-2019

Observa que hemos de definido para la tabla **alumno** y **materia** los campos que serán *llaves primarias* (PRIMARY KEY) inmediatamente después definir el tipo de dato y si permitirá almacenar datos Nulos o no (agregando la sintaxis NOT NULL o simplemente obviándola para el caso contrario). Sin embargo, para definir que más de un campo formara parte de la PRIMARY KEY de la tabla (como vemos en la tabla **nota**), agregamos al final de la declaración de todos los campos la sintaxis PRIMARY KEY(campos1, campos2,...).

Como también se observan las llaves foráneas (FOREIGN KEY) de la tabla **nota**. SQLite soporta la declaración de dichas llaves foráneas, pero no las toma en cuenta.

Ahora escribe el código anterior dentro del área de texto en la pestaña **Consulta SQL** y luego presiona el botón **ejecutar consulta**  Veras algo como esto:



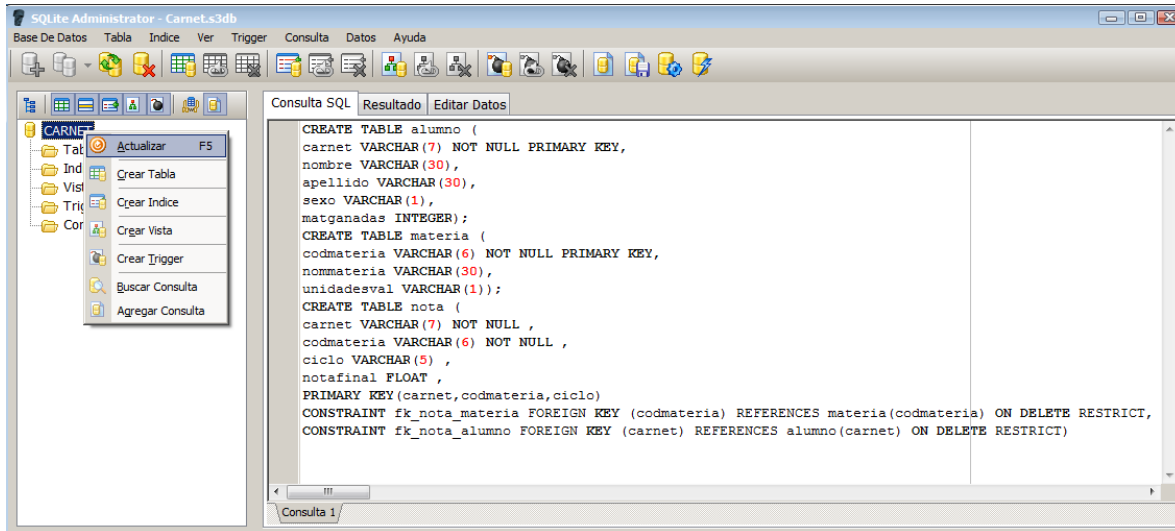
```
CREATE TABLE alumno (
carnet VARCHAR (7) NOT NULL PRIMARY KEY,
nombre VARCHAR (30),
apellido VARCHAR (30),
sexo VARCHAR (1),
matricadas INTEGER);
CREATE TABLE materia (
codmateria VARCHAR (6) NOT NULL PRIMARY KEY,
nommateria VARCHAR (30),
unidadesval VARCHAR (1);
CREATE TABLE nota (
carnet VARCHAR (7) NOT NULL ,
codmateria VARCHAR (6) NOT NULL ,
ciclo VARCHAR (5) ,
notafinal FLOAT ,
PRIMARY KEY (carnet,codmateria,ciclo)
CONSTRAINT fk_nota_materia FOREIGN KEY (codmateria) REFERENCES materia(codmateria) ON DELETE RESTRICT,
CONSTRAINT fk_nota_alumno FOREIGN KEY (carnet) REFERENCES alumno(carnet) ON DELETE RESTRICT)
```


Ahora presiona **F5** para actualizar el contenido de la base de datos ALUMNO, o bien da click derecho sobre la base de datos ALUMNO y selecciona ACTUALIZAR.

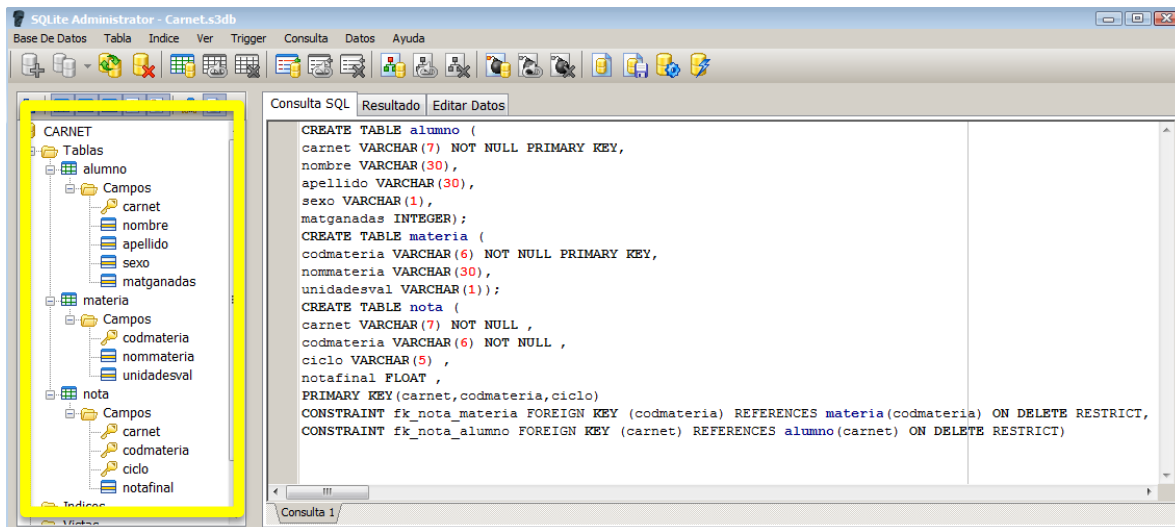



UNIVERSIDAD DE EL SALVADOR
FACULTAD DE INGENIERIA Y ARQUITECTURA
ESCUELA DE INGENIERIA DE SISTEMAS INFORMATICOS
PROGRAMACION PARA DISPOSITIVOS MOVILES
PDM115

Ciclo I-2019



Podrás explorar las tablas y los campos que están definidos en ellas, así como los campos que son PRIMARY KEY de la tabla, mostrados por un icono de llave 

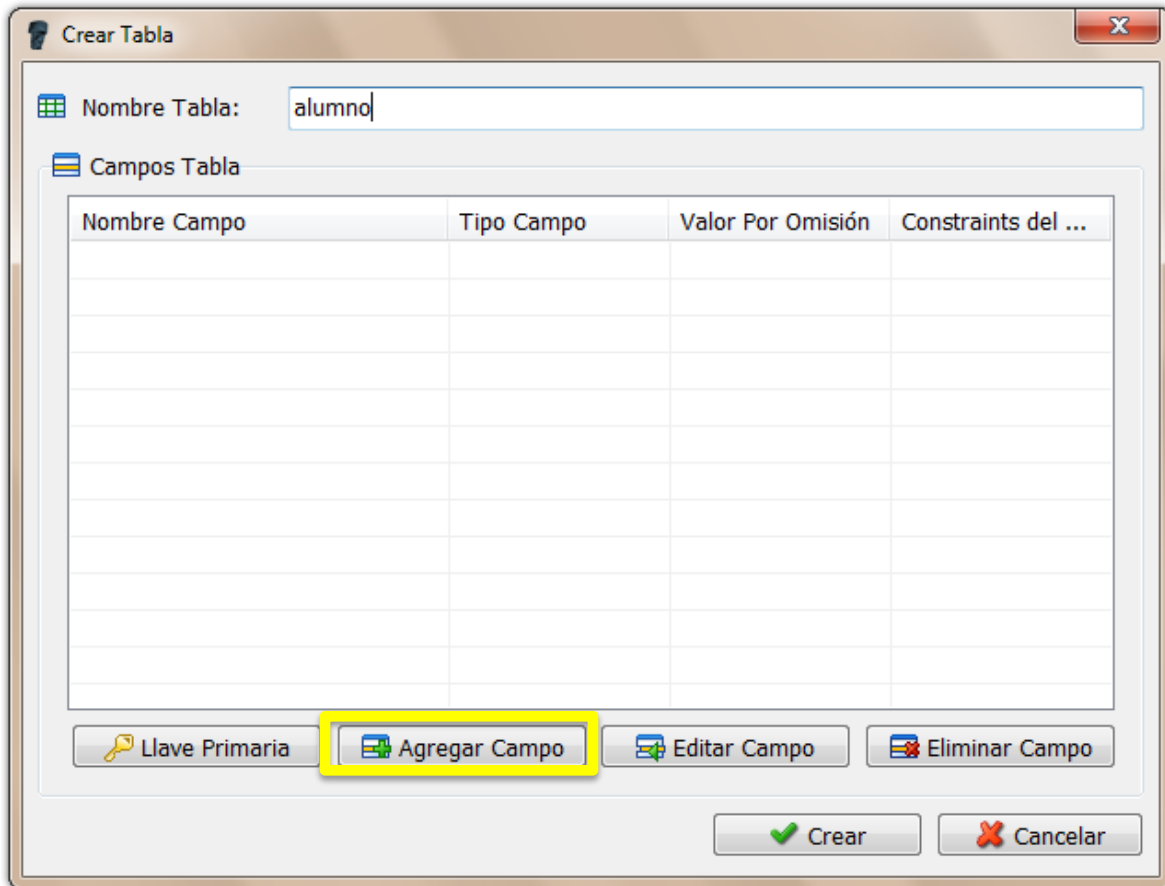


Otra manera para crear tablas, es mediante el uso del asistente integrado de SQLite Administrator, regresando a un estado inicial de la base de datos **carnet.s3db**, ve al menú **Tabla -> Nuevo**, o bien puedes hacer click sobre el botón de acceso rápido **Crear Tabla** 

Eliminaremos la tabla alumno, para crearla con el asistente, y observaremos que de igual manera se crea la tabla tanto con el asistente como con el código sql.



Veras la ventana del asistente Crear Tabla, en el campo **Nombre Tabla** escribe *alumno*.
Ahora presiona el botón **Agregar Campo**.



Agregaremos ahora un nuevo campo a la tabla; en Nombre Campo escribe **carnet**, en Tipo de Campo selecciona VARCHAR, marca que este campo es **Llave Primaria**, y que es **No NULL**. Presiona el botón **Agregar** y veras una nueva ventana que pide el tamaño del campo. Ingresa para para este caso un valor de 7 y luego click en **OK**.



Agregar Campo

Nombre Campo: carnet

Tipo Campo: VARCHAR

Valor por Omisión

Llave Primaria Unico

Autoincrementar No NULL

✓ Agregar ✗ Cancelar

Tamaño Campo:

Indique el Tamaño del Campo:

7

OK Cancel



Haremos lo mismo para los demás campos de la tabla, de la siguiente manera:

Dialog box titled "Agregar Campo" with the following fields and options:

- Nombre Campo: nombre
- Tipo Campo: VARCHAR
- Valor por Omisión: (empty)
- Llave Primaria
- Unico
- Autoincrementar
- No NULL

Buttons:

Dialog box titled "Tamaño Campo" with the following field and options:

- Indique el Tamaño del Campo: 30

Buttons:

Dialog box titled "Agregar Campo" with the following fields and options:

- Nombre Campo: apellido
- Tipo Campo: VARCHAR
- Valor por Omisión: (empty)
- Llave Primaria
- Unico
- Autoincrementar
- No NULL

Buttons:

Dialog box titled "Tamaño Campo" with the following field and options:

- Indique el Tamaño del Campo: 30

Buttons:

Dialog box titled "Agregar Campo" with the following fields and options:

- Nombre Campo: sexo
- Tipo Campo: VARCHAR
- Valor por Omisión: (empty)
- Llave Primaria
- Unico
- Autoincrementar
- No NULL

Buttons:

Dialog box titled "Tamaño Campo" with the following field and options:

- Indique el Tamaño del Campo: 1

Buttons:



Nombre Campo: matganadas
Tipo Campo: INTEGER
Valor por Omisión:
 Llave Primaria Unico
 Autoincrementar No NULL
Agregar Cancelar

Ahora en la ventana **Crear Tabla** presiona el botón **Crear**.

Nombre Campo	Tipo Campo	Valor Por Omisión	Constraints del ...
carnet	VARCHAR(7)		NOT NULL PRIM...
nombre	VARCHAR(30)		NULL
apellido	VARCHAR(30)		NULL
sexo	VARCHAR(1)		NULL
matganadas	INTEGER		NULL

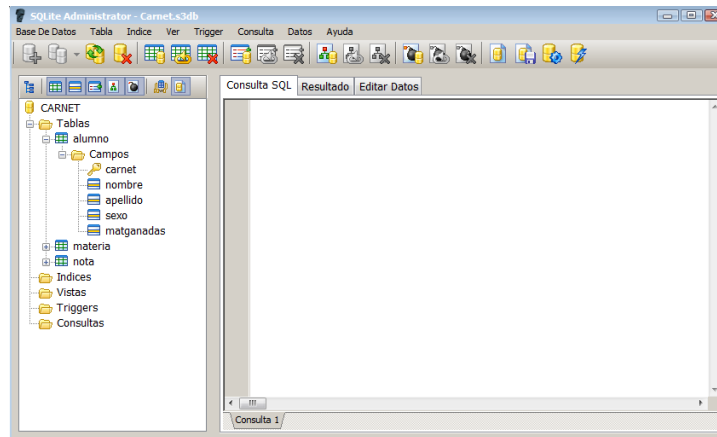
Llave Primaria Agregar Campo Editar Campo Eliminar Campo
Crear Cancelar

Con esto la tabla **alumno** ha sido creada sin utilizar comandos SQLite.



UNIVERSIDAD DE EL SALVADOR
FACULTAD DE INGENIERIA Y ARQUITECTURA
ESCUELA DE INGENIERIA DE SISTEMAS INFORMATICOS
PROGRAMACION PARA DISPOSITIVOS MOVILES
PDM115

Ciclo I-2019



INSERTAR REGISTROS

Ahora que tenemos nuestra base de datos creada con todas las tablas, empezaremos a llenarlas con datos, para ello, usaremos la siguiente sintaxis básica

INSERT INTO *nombreTabla*(*campo1, campo2,...*) **VALUES** (*dato_campo1,dato_campo2,...*)

Donde los valores (**VALUES**) están dispuestos en el mismo orden que los campos definidos después de **INSERT INTO**.

Ahora, escribe en **Consulta SQL** el siguiente código y ejecuta la consulta de la misma manera. Recuerda que siempre será necesario que una tabla “PADRE” contenga datos antes de insertar datos en una tabla “HIJO”, para este caso los registros que insertaremos en la tabla **nota** deben contener en los campos (que según el modelo físico poseen llaves foráneas) **carnet y codmateria**, solamente valores de **carnet** que posea la tabla **alumno** y valores de **codmateria** que posea la tabla **materia**. Por lo que con esto aclaramos que siempre debes insertar en tablas padre primero por la **integridad referencial**. **NOTA:** Datos alfanuméricos deben ingresarse entre comillas simples, datos numéricos no. El campo **codmateria** será un campo autoincrementable que explicaremos como darle esta función más adelante en la creación de **TRIGGERS** por lo que en los siguientes Insert solamente le asignaremos cero.

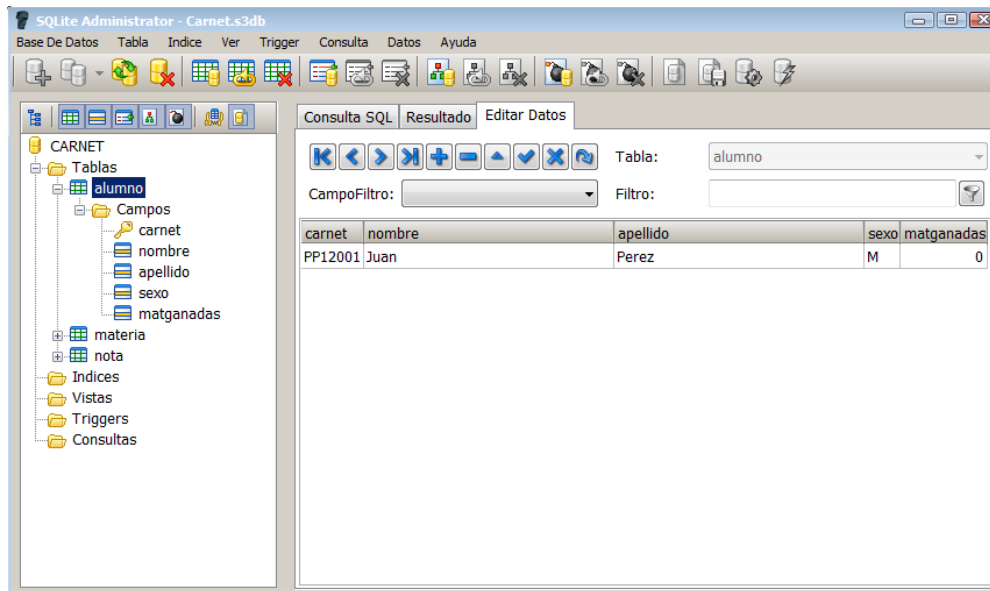
INSERT INTO alumno(carnet,nombre,apellido,sexo,matganadas) **VALUES** ('PP12001','Juan','Perez','M',0);

Después de ejecutar este comando, da click en la pestaña **Editar Datos**, y luego sobre la tabla **alumno**, observarás que el dato ha sido ingresado con éxito.



UNIVERSIDAD DE EL SALVADOR
 FACULTAD DE INGENIERIA Y ARQUITECTURA
 ESCUELA DE INGENIERIA DE SISTEMAS INFORMATICOS
 PROGRAMACION PARA DISPOSITIVOS MOVILES
 PDM115

Ciclo I-2019



Puedes ejecutar más de un comando INSERT a la vez, solamente debes indicar con un “;” donde es que finaliza cada uno.

```

Consulta SQL | Resultado | Editar Datos
INSERT INTO alumno (carnet,nombre,apellido,sexo,matganadas) VALUES ('0012035','Carlos','Orantes','M',2);
INSERT INTO alumno (carnet,nombre,apellido,sexo,matganadas) VALUES ('OF12044','Pedro','Ortiz','M',1);
INSERT INTO alumno (carnet,nombre,apellido,sexo,matganadas) VALUES ('GG11098','Sara','Gonzales','F',0);
INSERT INTO alumno (carnet,nombre,apellido,sexo,matganadas) VALUES ('CC12021','Gabriela','Coto','F',0);
  
```

Ahora inserta los siguientes datos en la tabla que corresponda.

Alumno				
carnet	nombre	apellido	sexo	matganadas
OO12035	Carlos	Orantes	M	2
OF12044	Pedro	Ortiz	M	1
GG11098	Sara	Gonzales	F	0
CC12021	Gabriela	Coto	F	0

Materia		
codmateria	nommateria	unidadesval
MAT115	Matematicas	4
PRN115	Programacion I	4



UNIVERSIDAD DE EL SALVADOR
 FACULTAD DE INGENIERIA Y ARQUITECTURA
 ESCUELA DE INGENIERIA DE SISTEMAS INFORMATICOS
 PROGRAMACION PARA DISPOSITIVOS MOVILES
 PDM115

Ciclo I-2019

IEC115	Ingenieria Economica	4
TSI115	Teoria de Sistemas	4

Nota			
carnet	codmateria	ciclo	notafinal
OO12035	MAT115	1	7
OF12044	PRN115	1	5
GG11098	IEC115	2	8
CC12021	TSI115	2	9
OO12035	IEC115	2	6
GG11098	MAT115	1	10
OF12044	PRN115	2	7

Observe las seis filas resaltadas, si bien sabemos que una llave primaria debe ser única, lo que significa que no puede repetirse su valor en otro registro de la misma tabla, en este caso vemos que carnet se repite en ambas filas, sin embargo, la **PRIMARY KEY** es el conjunto de los tres campos (carnet, codmateria y ciclo), por lo que la combinación de ambas es la que no se puede repetir. Por ejemplo para el registro resaltado en amarillo, tanto **carnet** como **codmateria** se repiten en ambos, sin embargo ciclo es distinto, por lo que sigue siendo válido.



MODIFICAR REGISTROS

Para actualizar/modificar registros de una tabla, usaremos la siguiente sintaxis básica:

UPDATE nombreTabla **SET** campo1=valorNuevo, campo2=valorNuevo2,... **WHERE** condición

Por ejemplo si quisiéramos modificar **notafinal** del registro de la tabla **nota**:

CC12021	TSI115	2	9
---------	--------	---	---

Por un **notafinal = 7**

CC12021	TSI115	2	7
---------	--------	---	---

Debemos ejecutar el siguiente código:

```
UPDATE nota SET notafinal=7 WHERE carnet='CC12021' AND codmateria='TSI115' AND ciclo=2
```

Observe que con la cláusula WHERE especificamos que no queremos actualizar el valor de todos los registros de la columna **notafinal**, sino únicamente la fila (o filas) que contengan los valores de **carnet**, **codmateria** y **ciclo** especificados.

Notaras que efectivamente el registro ha sido modificado por el nuevo valor:

The screenshot shows the SQLite Administrator interface for a database named 'CARNET.s3db'. The left sidebar displays the database structure with tables: alumno, materia, and nota. The main window shows the 'nota' table with the following data:

carnet	codmateria	ciclo	notafinal
OO12035	MAT115	1	7
OF12044	PRN115	1	5
GG11098	IJC115	2	8
CC12021	TSI115	2	7
OO12035	IJC115	2	6
GG11098	MAT115	1	10
OF12044	PRN115	2	7

The record with carnet 'CC12021', codmateria 'TSI115', ciclo '2', and notafinal '7' is highlighted with a yellow background, indicating it has been successfully updated.




Recuerda que debes respetar la integridad "PADRE-HIJO" por lo que si quisieras modificar un registro de la tabla PADRE, que hace referencia a registros de la tabla HIJO, deberás modificar ambos. Por ejemplo, si quisieras modificar un carnet de la tabla **alumno**, y dicho carnet posee **nota**, entonces deberás modificar el carnet de ambas tablas.

CONSULTAR REGISTROS

Para consultar registros de las tablas usaremos la siguiente sintaxis:

```
SELECT campo1,campo2,.. FROM tabla1,tabla2,... WHERE condición
```

Por ejemplo ejecuta el siguiente comando, pero esta vez presiona el botón **Ejecutar consulta SQL con Resultado**  donde le pediremos a la base de datos que nos muestre la información del alumno con carnet *OF12044* y todas las notas relacionadas con él.

```
SELECT alumno.carnet,nombre,apellido,codmateria,ciclo,notafinal FROM alumno,nota  
WHERE alumno.carnet='OF12044'AND alumno.carnet=nota.carnet;
```

Observa que el campo carnet lo poseen ambas tablas, por lo que para hacer referencia al campo de una tabla específica podemos usar el nombre de la tabla seguido de un punto y el nombre de uno de sus campos. (**TABLA.CAMPON**), además usamos **alumno.carnet=nota.carnet** para evitar que muestre registros repetidos.

El resultado de esta consulta es el siguiente.

carnet	nombre	apellido	codmateria	ciclo	notafinal
OF12044	Pedro	Ortiz	PRN115	1	5
OF12044	Pedro	Ortiz	PRN115	2	7



ELIMINAR REGISTROS

Para eliminar registros de la base de datos, usaremos la siguiente sintaxis básica.

DELETE FROM nombreTabla **WHERE** condición

Eliminaremos el primer registro que insertamos ('PP12001','Juan','Perez','M'), por lo ejecutaremos el siguiente comando.

```
DELETE FROM alumno WHERE carnet='PP12001';
```

Ya que este Alumno no posee notas, no será necesario eliminar registros de la tabla **nota** antes de eliminar registro de la tabla **alumno**, por lo que lo eliminamos sin ninguna complicación. Observaras que efectivamente el registro ha sido eliminado.

carnet	nombre	apellido	sexo	matganadas
0012035	Carlos	Orantes	M	2
OF12044	Pedro	Ortiz	M	1
GG11098	Sara	Gonzales	F	0
CC12021	Gabriela	Coto	F	0

Sin embargo si quisiéramos eliminar de la tabla **alumno** el registro con carnet *0012035*, debemos eliminar todos los registros que posean este mismo valor de carnet en la tabla **nota**, y posteriormente eliminar de la tabla **alumno**.



TRIGGERS

Un trigger (o disparador) en una Base de datos, es un procedimiento que se ejecuta cuando se cumple una condición establecida al realizar una operación. Dependiendo de la base de datos, los triggers pueden ser de inserción (insert), actualización (update) o borrado (delete).

Sus usos y características son:

- Mejoran la administración de la Base de datos, sin necesidad de contar con que el usuario ejecute la sentencia de SQL.
- Pueden generar valores de columnas
- Previene errores de datos
- Sincroniza tablas
- Modifica valores de una vista
- No aceptan parámetros o argumentos (pero podrían almacenar los datos afectados en tablas temporales)
- No pueden ejecutar las operaciones *COMMIT* o *ROLLBACK* por que estas son parte de la sentencia SQL del disparador (únicamente a través de transacciones autónomas)

Componentes principales

- **Llamada de activación:** es la sentencia que permite "disparar" el código a ejecutar.
- **Restricción:** es la condición necesaria para realizar el código. Esta restricción puede ser de tipo condicional o de tipo nulidad.
- **Acción a ejecutar:** es la secuencia de instrucciones a ejecutar una vez que se han cumplido las condiciones iniciales.

Trigger en SQLite

Consideraciones al momento de realizar u operar un trigger:

- La sentencia CREATE TRIGGER se utiliza para añadir mecanismos de activación para el esquema de la base de datos. Los factores desencadenantes son las operaciones de base de datos que se realizan automáticamente cuando un evento ocurre en la base de datos especificada.
- Un disparador puede ser especificado para disparar cada vez que un DELETE, INSERT, o UPDATE de una tabla de base de datos determinada se produce, o cuando se produce una actualización de una o varias columnas especificadas de una tabla.
- Tanto la cláusula WHEN y las acciones del trigger puede acceder a los elementos de la fila que se inserta, elimina o actualiza con las referencias de la forma "NEW.nombre_columna" y "OLD.nombre_columna", donde nombre_columna es el nombre de una columna de la tabla con la que el trigger está asociado. Las referencias OLD y NEW solo pueden utilizarse en los triggers o los eventos en los cuales son relevantes, de esta manera:
 - INSERT, son válidas las referencias NEW
 - UPDATE, son válidas las referencias NEW y OLD
 - BORRAR, son válidas las referencias OLD
- Los triggers se eliminan automáticamente cuando la tabla a la que están asociados se elimina. Sin embargo, si las acciones de activación referencia a otras tablas, el trigger no se elimina o modifica si esas otras tablas se eliminan o modifican.



Un ejemplo del uso de Triggers

Suponiendo que los registros de clientes se almacenan en la tabla "customers", y que los registros de pedidos se almacenan en la tabla "orders", el siguiente trigger se asegura de que todos los pedidos asociados se redirigen cuando un cliente cambia su dirección:

```
CREATE TRIGGER update_customer_address UPDATE OF address ON customers
BEGIN
    UPDATE orders SET address = new.address WHERE customer_name =
old.name;
END;
```

Con este disparador instalado, al ejecutar el comando:

```
UPDATE customers SET address = '1 Main St.' WHERE name = 'Jack Jones';
```

Provoca que el siguiente UPDATE se ejecute automáticamente el que se define en el TRIGGER):

```
UPDATE orders SET address = '1 Main St.' WHERE customer_name =
'Jack Jones';
```

Continuando con la práctica...

Como vemos, un trigger nos permitirá ejecutar comandos SQL (INSERT, DELETE, UPDATE) cada vez que la condición con la que se ha creado se cumpla.

matganadas de la tabla **alumno** representa las materias ganadas (aprobadas) por un alumno, por lo que debemos llevar el control de todas las materias "ganadas" por este alumno de forma automática, para lograr esto ocuparemos un trigger que se activara de la siguiente manera: cuando se ingrese una **notafinal** de una materia en la tabla **nota** para un **alumno** específico, debemos verificar que dicha **notafinal** esté aprobada (nota mayor o igual a 6.0), y por lo tanto, formaría parte de las materias "ganadas" por el **alumno**, por lo que luego de verificar que cumpla la condición **notafinal >= 6.0** el valor del campo **matganadas** de dicho alumno deberá incrementarse para llevar el conteo de todas las materias ganadas en su record académico, por lo que debemos ejecutar un **UPDATE** del campo **matganadas** del **alumno** en cuestión.

Crearemos un trigger usando comandos SQLite, el código es el siguiente:



CREATE TRIGGER update_matganadas

```
AFTER INSERT ON nota
WHEN new.notafinal>=6
BEGIN

UPDATE alumno SET matganadas=matganadas+1 WHERE alumno.carnet=new.carnet;

END
```

Puedes ver más acerca de la sintaxis de creación de un trigger en la pag.

http://www.sqlite.org/lang_createttrigger.html

CREATE TRIGGER *nombre_del_trigger*: con esto indicamos la creación de un trigger que tendrá como nombre *nombre_del_trigger*.

AFTER. Para indicar que el trigger se activara después de un evento indicado.

INSERT Evento que causará la activación del trigger (siempre que se cumpla la condición si es que se define una).

ON *nombre_tabla*. Además de haber definido el evento que activara el trigger, debemos indicar la tabla, de tal manera que en este caso, estamos definiendo que el trigger se activara cada vez que exista un INSERT dentro de la tabla *nombre_tabla*.

WHEN *condición*. La condición para que la acción definida dentro de **BEGIN...END** del trigger se ejecute. Para este caso usamos la referencia NEW, con lo que decimos que si el valor *NUEVO* (el que se está insertando) del campo *notafinal* es mayor o igual a 6 entonces ejecutar la acción...

BEGIN o empezar en español, indica donde definiremos los comandos SQL que queremos que se ejecuten como respuesta de la activación del trigger ya que se ha cumplido la condición dada.

Para este caso vemos que realizamos un UPDATE de la tabla *alumno*, en el que le asignamos un nuevo valor al campo *matganadas*, que es igual a incrementar en 1 el valor anterior, solo para el *carnet* del alumno que sea igual al *carnet* del registro *NUEVO* que se está insertando.

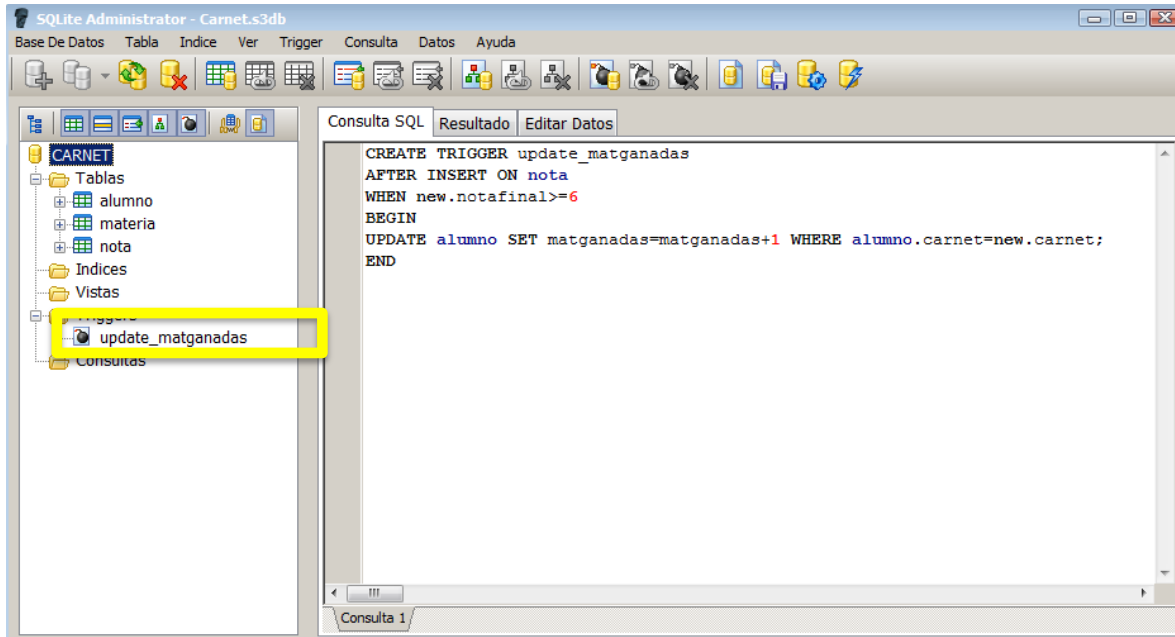
END, fin de los comandos que se ejecutarán.

Ahora ejecuta el código anterior dentro de la pestaña **Consulta SQL** de SQLite Administrator. Verás que como resultado se creara un nuevo elemento dentro de la carpeta TRIGGER de esta manera:



UNIVERSIDAD DE EL SALVADOR
FACULTAD DE INGENIERIA Y ARQUITECTURA
ESCUELA DE INGENIERIA DE SISTEMAS INFORMATICOS
PROGRAMACION PARA DISPOSITIVOS MOVILES
PDM115

Ciclo I-2019



Ahora haremos una prueba.

Inserta un nuevo alumno:

```
INSERT INTO alumno VALUES ('NN00001','Nuevo','Nuevo','M',0);
```

Tendremos ahora en la tabla **alumno** los datos:

carnet	nombre	apellido	sexo	matganadas
OO12035	Carlos	Orantes	M	0
OF12044	Pedro	Ortiz	M	0
GG11098	Sara	Gonzales	F	0
CC12021	Gabriela	Coto	F	0
NN00001	Nuevo	Nuevo	M	0

Y Ahora inserta un registro en la tabla **nota** para el alumno nuevo 'NN00001'

```
INSERT INTO nota VALUES('NN00001','MAT115','1',8);
```

Al ver los datos de la tabla **alumno**, el campo **matganadas** del alumno 'NN00001' se ha incrementado en 1 tal y como esperábamos mediante el trigger.

carnet	codmateria	ciclo	notafinal
NN00001	MAT115	1	8

carnet	nombre	apellido	sexo	matganadas
OO12035	Carlos	Orantes	M	0
OF12044	Pedro	Ortiz	M	0
GG11098	Sara	Gonzales	F	0
CC12021	Gabriela	Coto	F	0
NN00001	Nuevo	Nuevo	M	1



UNIVERSIDAD DE EL SALVADOR
FACULTAD DE INGENIERIA Y ARQUITECTURA
ESCUELA DE INGENIERIA DE SISTEMAS INFORMATICOS
PROGRAMACION PARA DISPOSITIVOS MOVILES
PDM115

Ciclo I-2019



Ahora inserta una materia reprobada para el mismo alumno.

```
INSERT INTO nota VALUES('NN00001','IEC115','1',5.5);
```

Y veremos que el valor de **matganadas** no ha sido modificado, puesto que 5.5 no es mayor a 6 según la condición del trigger.

carnet	codmateria	ciclo	notafinal
NN00001	MAT115	1	8
NN00001	IEC115	1	5.5

carnet	nombre	apellido	sexo	matganadas
OO12035	Carlos	Orantes	M	0
OF12044	Pedro	Ortiz	M	0
GG11098	Sara	Gonzales	F	0
CC12021	Gabriela	Coto	F	0
NN00001	Nuevo	Nuevo	M	1

Sin embargo, en caso de que la **notafinal** se guardó con un valor incorrecto y posteriormente deba modificarse, debemos tener el control de esto usando otro trigger que permita decrementar el valor de **matganadas** si **notafinal** se modificara a un valor menor de 6, o que se incremente en caso contrario.

Para ello crearemos otros 2 triggers que tendrán el control del evento UPDATE solamente cuando se modifique el valor del campo **notafinal** sobre la tabla **nota**.

Ejecuta primero este código.

```
CREATE TRIGGER corregir_notafinal_aprobada  
AFTER UPDATE OF notafinal ON nota  
FOR EACH ROW WHEN new.notafinal>=6 AND old.notafinal<6  
BEGIN  
UPDATE alumno SET matganadas=matganadas+1 WHERE alumno.carnet=new.carnet;  
END
```

Ahora ejecuta este otro.

```
CREATE TRIGGER corregir_notafinal_noaprobada
```



UNIVERSIDAD DE EL SALVADOR
 FACULTAD DE INGENIERIA Y ARQUITECTURA
 ESCUELA DE INGENIERIA DE SISTEMAS INFORMATICOS
 PROGRAMACION PARA DISPOSITIVOS MOVILES
 PDM115

Ciclo I-2019

```
AFTER UPDATE OF notafinal ON nota
FOR EACH ROW WHEN new.notafinal<6 AND old.notafinal>=6
BEGIN
UPDATE alumno SET matganadas=matganadas-1 WHERE alumno.carnet=new.carnet;
END
```

Ahora actualizemos el registro de la materia reprobada con 5.5 por un valor de 6.0.

NN00001	MAT115	1	8
NN00001	IEC115	1	6

```
UPDATE nota SET notafinal=6 WHERE carnet= 'NN00001' AND codmateria='IEC115' AND ciclo='1';
```

Con lo que el campo **matganadas** incrementa a 2.

carnet	codmateria	ciclo	notafinal
--------	------------	-------	-----------

carnet	nombre	apellido	sexo	matganadas
OO12035	Carlos	Orantes	M	0
OF12044	Pedro	Ortiz	M	0
GG11098	Sara	Gonzales	F	0
CC12021	Gabriela	Coto	F	0
NN00001	Nuevo	Nuevo	M	2

Y si hacemos lo contrario para la materia anteriormente aprobada con 8.

```
UPDATE nota SET notafinal=3 WHERE carnet= 'NN00001' AND codmateria='MAT115' AND ciclo='1';
```

Tendremos el siguiente resultado:

carnet	codmateria	ciclo	notafinal
NN00001	MAT115	1	3
NN00001	IEC115	1	6

carnet	nombre	apellido	sexo	matganadas
OO12035	Carlos	Orantes	M	0
OF12044	Pedro	Ortiz	M	0
GG11098	Sara	Gonzales	F	0
CC12021	Gabriela	Coto	F	0
NN00001	Nuevo	Nuevo	M	1



RESTRICCION DE INTEGRIDAD REFERENCIAL(Foreign Key)

Como observamos inicialmente, creamos unas llaves foráneas correspondientes a la tabla **nota** y se mencionó que SQLite ignora esas llaves foráneas creadas, por lo tanto la integridad relacional no se respeta.

Podremos observar que al agregar una nota a un estudiante que no existe, SQLite nos permitirá realizar esta inserción, de igual manera si ingresáramos una nota con referencia a una materia que no está registrada. Para corroborar ingresar el siguiente registro en la tabla nota:

```
INSERT INTO nota VALUES('BB01007','PRN215','1',8);
```

Podremos observar que la inserción se realiza correctamente, sin importar que tanto el alumno como la materia no se encuentren registrados en la base de datos. Para evitar estos errores se deben de realizar triggers que controlen la inserción de datos a las tablas que contengan llaves foráneas. Eliminamos el registro anteriormente ingresado.

Crearemos 2 triggers diferentes para la verificación de existencia de los registros que se desean ingresar, uno para la verificación de carnet de alumno y el otro para la verificación del código de la materia de la materia a la cual le corresponde la nota.

Ejecutar el siguiente código:

```
CREATE TRIGGER fk_nota_alumno
BEFORE INSERT ON nota
FOR EACH ROW
BEGIN
    SELECT CASE
        WHEN ((SELECT carnet FROM alumno WHERE carnet = NEW.carnet) IS NULL)
        THEN RAISE(ABORT, 'No existe alumno')
    END;
END;
```



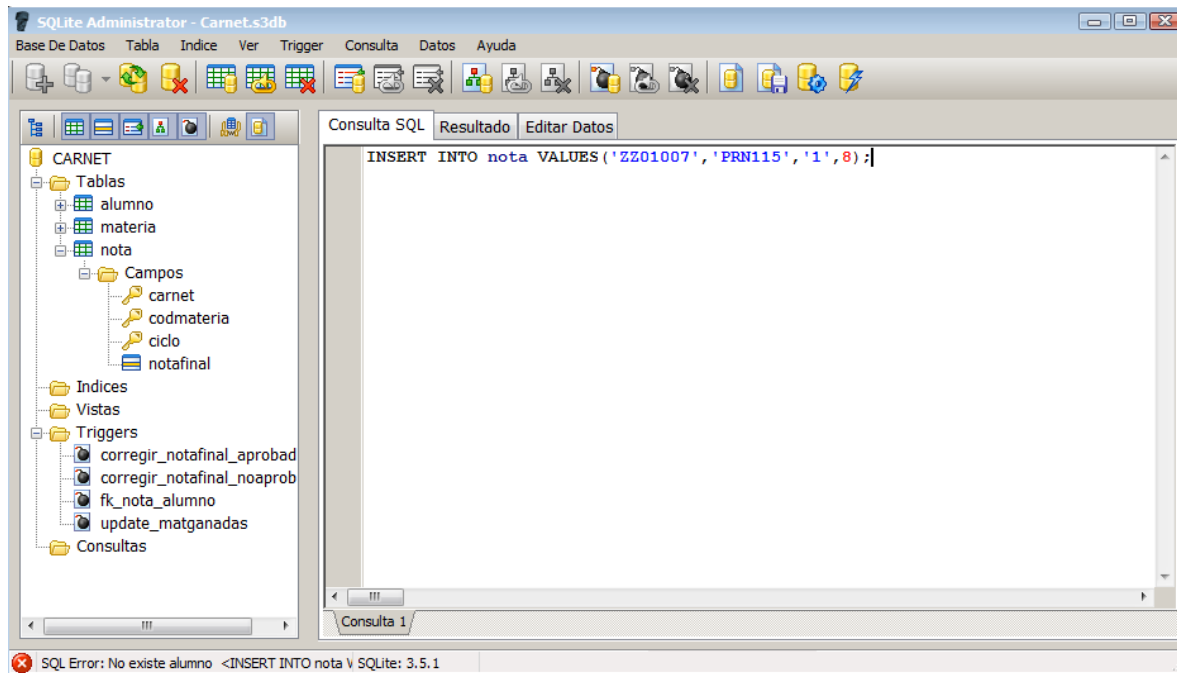
UNIVERSIDAD DE EL SALVADOR
FACULTAD DE INGENIERIA Y ARQUITECTURA
ESCUELA DE INGENIERIA DE SISTEMAS INFORMATICOS
PROGRAMACION PARA DISPOSITIVOS MOVILES
PDM115

Ciclo I-2019

Creado el trigger anterior, intentamos insertar una nota de un alumno que no existe en la base de datos, ingresemos la sentencia que se presenta a continuación.

```
INSERT INTO nota VALUES('ZZ01007','PRN115','1',8);
```

Observamos que se presenta un error SQL, se puede observar en la parte inferior de la pantalla, con el texto que se ha ingresado en el trigger, en este caso 'No existe alumno', y correctamente se evita la inserción de este registro.



De igual manera para realizar el trigger de la llave foránea de nota y materia, ejecutar el siguiente código:

```
CREATE TRIGGER fk_nota_materia  
BEFORE INSERT ON nota  
FOR EACH ROW  
BEGIN  
    SELECT CASE
```



UNIVERSIDAD DE EL SALVADOR
FACULTAD DE INGENIERIA Y ARQUITECTURA
ESCUELA DE INGENIERIA DE SISTEMAS INFORMATICOS
PROGRAMACION PARA DISPOSITIVOS MOVILES
PDM115

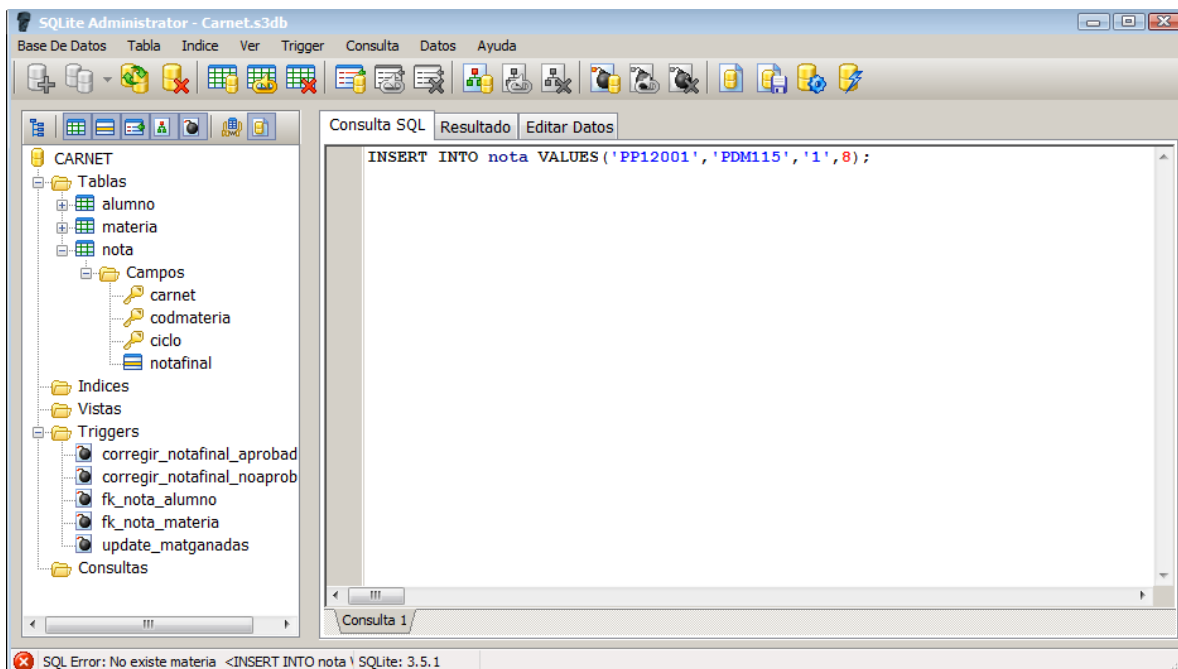
Ciclo I-2019

```
WHEN ((SELECT codmateria FROM materia WHERE codmateria = NEW.codmateria) IS NULL)
    THEN RAISE(ABORT, 'No existe materia')
END;
END;
```

Creado el trigger anterior, intentamos insertar una nota de una materia que no existe en la base de datos, ingresemos la sentencia que se presenta a continuación.

```
INSERT INTO nota VALUES('PP12001','PDM115','1',8);
```

Observamos que se presenta un error igual en SQL, se puede observar en la parte inferior de la pantalla, con el texto que se ha ingresado en el trigger, en este caso 'No existe materia', y correctamente se evita la insercion de este registro.





Anexo

Modelo Conceptual de esquema Carnet.

