Chapter 21: Database Programming

This chapter will show how BASIC-256 can connect to a simple relational database and use it to store and retrieve useful information.

What is a Database:

A database is simply an organized collection of numbers, string, and other types of information. The most common type of database is the "Relational Database". Relational Databases are made up of four major parts: tables, rows, columns, and relationships (see Table 9).

Table	A table consists of a predefined number or columns any any number of rows with information about a specific object or subject. Also known as a relation.
Row	Also called a tuple.
Column	This can also be referred to as an attribute.
Relationship	A reference of the key of one table as a column of another table. This creates a connection between tables.

Table 9: Major Components of a Relational Database

The SQL Language:

Most relational databases, today, use a language called SQL to actually extract and manipulate data. SQL is actually an acronym for Structured Query Language. The original SQL language was developed by IBM in the 1970s and has become the primary language used by relational databases.

SQL is a very powerful language and has been implemented by dozens of software companies, over the years. Because of this complexity there are many different dialects of SQL in use. BASIC-256 uses the SQLite database engine. Please see the SQLite web-page at http://www.sqlite.org for more information about the dialect of SQL shown in these examples.

Creating and Adding Data to a Database:

The SQLite library does not require the installation of a database sever or the setting up of a complex system. The database and all of its parts are stored in a simple file on your computer. This file can even be copied to another computer and used, without problem.

The first program (Program 127: Create a Database) creates a new sample database file and tables. The tables are represented by the Entity Relationship Diagram (ERD) as shown in Illustration 39.

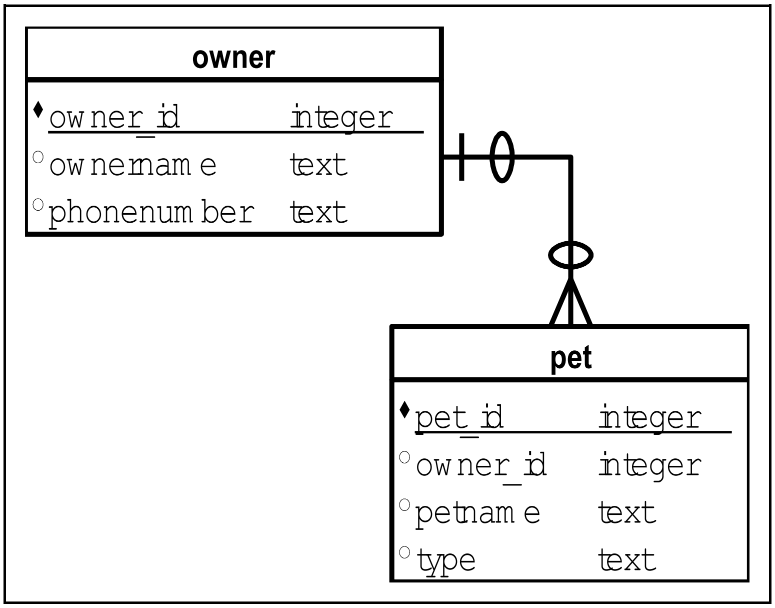

Illustration 39: Entity Relationship Diagram of Chapter Database

```
# dbcreate.kbs - create the pets database and tables

# delete old database and create a database with two tables
file = "pets.sqlite3"
if exists(file) then kill(file)
dbopen file

stmt =  "CREATE TABLE owner (owner_id INTEGER, ownername TEXT, phonenumber TEXT, PRIMARY KEY (owner_id));"
call executeSQL(stmt)

stmt =  "CREATE TABLE pet (pet_id INTEGER, owner_id INTEGER, petname TEXT, type TEXT, PRIMARY KEY (pet_id), FOREIGN KEY (owner_id) REFERENCES owner (owner_id));"
call executeSQL(stmt)

# wrap everything up
dbclose
print file + " created."
end

subroutine executeSQL(stmt)
     print stmt
     try
          dbexecute stmt
     catch
          print "Caught Error"
          print "   Error = " + lasterror
          print "   On Line = " + lasterrorline
          print "   Message = " + lasterrormessage
     endtry
end subroutine
```

Program 127: Create a Database

```
CREATE TABLE owner (owner_id INTEGER, ownername
TEXT, phonenumber TEXT, PRIMARY KEY
(owner_id));
CREATE TABLE pet (pet_id INTEGER, owner_id
INTEGER, petname TEXT, type TEXT, PRIMARY KEY
(pet_id), FOREIGN KEY (owner_id) REFERENCES
owner (owner_id));
pets.sqlite3 created.
```

Sample Output 127: Create a Database

So far you have seen three new database statements: **dbopen** – will open a database file and create it if it does not exist, **dbexecute** – will execute an SQL statement on the open database, and **dbclose** – closes the open database file.

New Concept

```
dbopen filename
```

Open an SQLite database file. If the database does not exist then create a new empty database file.

New Concept

```
dbexecute sqlstatement
```

Perform the SQL statement on the currently open SQLite database file. No value will be returned but a trappable runtime error will occur if there were any problems executing the statement on the database.

New Concept

```
dbclose
```

Close the currently open SQLite database file. This statement insures that all data is written out to the database file.

These same three statements can also be used to execute other SQL statements. The INSERT INTO statement (Program 128) adds new rows of data to the tables and the UPDATE statement (Program 129) will change an existing row's information.

Warning	When you are building a SQL statement that may contain informtion typed in by the user, you must be very careful and handle quotation marks that they might type in. Malicious users may try to do something called an SQL-Injection where they will embed a harmful SQL statement into what they have entered into the program. Data may be lost or compromised if care is not taken. The following examples use a function called "quote" that will quote a string containing quotation marks correctly and should eliminate this risk for simple programs.

The "quote" function will place single quotation marks around a string and return the string with the quotes. If a string contains single quotations within it, they will be doubled and handled correctly by SQLite.

```
# quote.kbs - quote a string for SQLite
# SAVE IT AS quote.kbs
#
# wrap a string in single quotes (for a sql statement)
# if it contains a single quote double it
function quote(a)
    return "'" + replace(a,"'","''") + "'"
end function
```

```
# dbinsert.kbs - add rows to the database

include "quote.kbs"

file = "pets.sqlite3"
dbopen file

call addowner(1, "Jim", "555-3434")
call addpet(1, 1, "Spot", "Cat")
call addpet(2, 1, "Fred", "Cat")
call addpet(3, 1, "Elvis", "Cat")

call addowner(2, "Sue", "555-8764")
call addpet(4, 2, "Alfred", "Dog")
call addpet(5, 2, "Fido", "Cat")

call addowner(3, "Amy", "555-4321")
call addpet(6, 3, "Bones", "Dog")

call addowner(4, "Dee", "555-9659")
call addpet(7, 4, "Sam", "Goat")

# wrap everything up
dbclose
end

subroutine addowner(owner_id, ownername, phonenumber)
    stmt = "INSERT INTO owner (owner_id, ownername, phonenumber) VALUES (" + owner_id + "," + quote(ownername) + "," + quote(phonenumber) + ");"
    print stmt
    try
        dbexecute stmt
    catch
        print "Unbale to add owner " + owner_id + " " + lasterrorextra
    end try
end subroutine

subroutine addpet(pet_id, owner_id, petname, type)
    stmt = "INSERT INTO pet (pet_id, owner_id, petname, type) VALUES (" + pet_id + "," + owner_id + "," + quote(petname) + "," + quote(type) + ");"
    print stmt
    try
        dbexecute stmt
    catch
```

```
            print "Unbale to add pet " + pet_id + " " + lasterrorextra
      end try
endsubroutine
```

Program 128: Insert Rows into Database

```
INSERT INTO owner (owner_id, ownername, 
phonenumber) VALUES (1,'Jim','555-3434');
INSERT INTO pet (pet_id, owner_id, petname, 
type) VALUES (1,1,'Spot','Cat');
INSERT INTO pet (pet_id, owner_id, petname, 
type) VALUES (2,1,'Fred','Cat');
INSERT INTO pet (pet_id, owner_id, petname, 
type) VALUES (3,1,'Elvis','Cat');
INSERT INTO owner (owner_id, ownername, 
phonenumber) VALUES (2,'Sue','555-8764');
INSERT INTO pet (pet_id, owner_id, petname, 
type) VALUES (4,2,'Alfred','Dog');
INSERT INTO pet (pet_id, owner_id, petname, 
type) VALUES (5,2,'Fido','Cat');
INSERT INTO owner (owner_id, ownername, 
phonenumber) VALUES (3,'Amy','555-4321');
INSERT INTO pet (pet_id, owner_id, petname, 
type) VALUES (6,3,'Bones','Dog');
INSERT INTO owner (owner_id, ownername, 
phonenumber) VALUES (4,'Dee','555-9659');
INSERT INTO pet (pet_id, owner_id, petname, 
type) VALUES (7,4,'Sam','Goat');
```

Sample Output 128: Insert Rows into Database

```
# dbupdate.kbs - update a database row

include "quote.kbs"

dbopen "pets.sqlite3"
```

```
s$ =  "UPDATE owner SET phonenumber = " + quote("555-5555") + " where owner_id = 1;"
print s$
dbexecute s$
dbclose
```

Program 129: Update Row in a Database

```
UPDATE owner SET phonenumber = '555-5555' where
owner_id = 1;
```

Sample Output 129: Update Row in a Database

Retrieving Information from a Database:

So far we have seen how to open, close, and execute a SQL statement that does not return any values. A database would be pretty useless if we could not get information out of it.

The SELECT statement, in the SQL language, allows us to retrieve the desired data. After a SELECT is executed a "record set" is created that contains the rows and columns of data that was extracted from the database. Program 130 shows three different SELECT statements and how the data is read into your BASIC-256 program.

```
# showpetsdb.kbs
# display data from the pets database

dbopen "pets.sqlite3"

# show owners and their phone numbers
print "Owners and Phone Numbers"
dbopenset "SELECT ownername, phonenumber FROM owner ORDER BY ownername;"
while dbrow()
```

```
    print dbstring(0) + " " + dbstring(1)
end while
dbcloseset

print

# show owners and their pets
print "Owners with Pets"
dbopenset "SELECT owner.ownername, pet.pet_id, pet.petname, pet.type FROM owner JOIN pet ON pet.owner_id = owner.owner_id ORDER BY ownername, petname;"
while dbrow()
    print dbstring(0) + " " + dbint(1) + " " + dbstring(2) + " " + dbstring(3)
end while
dbcloseset

print

# show average number of pets
print "Average Number of Pets"
dbopenset "SELECT AVG(c) FROM (SELECT COUNT(*) AS c FROM owner JOIN pet ON pet.owner_id = owner.owner_id GROUP BY owner.owner_id) AS numpets;"
while dbrow()
    print dbfloat(0)
end while
dbcloseset

# wrap everything up
dbclose
```

Program 130: Selecting Sets of Data from a Database

```
Owners and Phone Numbers
Amy 555-9932
Dee 555-4433
```

```
Jim 555-5555
Sue 555-8764

Owners with Pets
Amy 6 Bones Dog
Dee 7 Sam Goat
Jim 3 Elvis Cat
Jim 2 Fred Cat
Jim 1 Spot Cat
Sue 4 Alfred Cat
Sue 5 Fido Dog

Average Number of Pets
1.75
```

Sample Output 130: Selecting Sets of Data from a Database

New Concept

```
dbopenset sqlstatement
```

Execute a SELECT statement on the database and create a "record set" to allow the program to read in the result. The "record set" may contain 0 or more rows as extracted by the SELECT.

New Concept

```
dbrow  or  dbrow ()
```

Function to advance the result of the last **dbopenset** to the next row. Returns false if we are at the end of the selected data.

You need to advance to the first row, using **dbrow**, after a **dbopenset** statement before you can read any data.

New Concept

```
dbint ( column )
dbfloat ( column )
dbstring ( column )
```

These functions will return data from the current row of the record set. You must know the zero based numeric column number of the desired data.

dbint	Return the cell data as an integer.
dbfloat	Return the cell data as a floating-point number.
dbstring	Return the cell data as a string.

New Concept

```
dbcloseset
```

Close and discard the results of the last **dbopenset** statement.

Big Program

The big program this chapter creates a single program that creates, maintains, and lists phone numbers stored in a database file.

Pay special attention to the quote function used in creating the SQL statements. It wraps all strings in the statements in single quotes after changing the single quotes in a string to a pair of them. This doubling of quotes inside quotes is how to insert a quotation mark in an SQL statement.

```
# rolofile.kbs
# a database example to keep track of phone numbers

include "quote.kbs"

dbopen "rolofile.sqlite3"
call createtables()

do
     print
     print "rolofile - phone numbers"
     print "1-add person"
     print "2-list people"
     print "3-add phone"
     print "4-list phones"
     input "0-exit >", choice
     print

     if choice=1 then call addperson()
     if choice=2 then call listpeople()
     if choice=3 then call addphone()
     if choice=4 then call listphone()
until choice = 0
dbclose
end

function inputphonetype()
```

```
    do
        input "Phone Type (h-home, c-cell, f-fax, w-work) > ", type
    until type = "h" or type = "c" or type = "f" or type = "w"
    return type
end function

subroutine createtables()
    # includes the IF NOT EXISTS clause to not error if the
    # table already exists
    dbexecute "CREATE TABLE IF NOT EXISTS person (person_id TEXT PRIMARY KEY, name TEXT);"
    dbexecute "CREATE TABLE IF NOT EXISTS phone (person_id TEXT, phone TEXT, type TEXT, PRIMARY KEY (person_id, phone));"
end subroutine

subroutine addperson()
    print "add person"
    input "person id > ", person_id
    person_id = upper(person_id)
    if ispersononfile(person_id) or person_id = "" then
        print "person already on file or empty"
    else
        inputstring "person name > ", person_name
        if person_name = "" then
            print "please enter name"
        else
            dbexecute "INSERT INTO person (person_id, name) VALUES (" + quote(person_id) + "," + quote(person_name) + ");"
            print person_id + " added."
        end if
    end if
end subroutine

```

```
subroutine addphone()
    print "add phone number"
    input "person id > ", person_id
    person_id = upper(person_id)
    if not ispersononfile(person_id) then
        print "person not on file"
    else
        inputstring "phone number > ", phone
        if phone = "" then
            print "please enter a phone number"
        else
            type = inputphonetype()
            dbexecute "INSERT INTO phone (person_id, phone, type) values (" + quote(person_id) + "," + quote(phone) + "," + quote(type) + ");"
            print phone + " added."
        end if
    end if
end subroutine

function ispersononfile(person_id)
    # return true/false whether the person is on the person table
    onfile = false
    dbopenset "select person_id from person where person_id = " + quote(person_id)
    if dbrow() then onfile = true
    dbcloseset
    return onfile
end function

subroutine listpeople()
    dbopenset "select person_id, name from person order by person_id"
    while dbrow()
        print dbstring("person_id") + " " + dbstring("name")
    end while
    dbcloseset
```

```
end subroutine

subroutine listphone()
     input "person id to list (return for all) > ", person_id
     person_id = upper(person_id)
     stmt = "SELECT person.person_id, person.name, phone.phone, phone.type FROM person LEFT JOIN phone ON person.person_id = phone.person_id"
     if person_id <> "" then stmt += " WHERE person.person_id = " + quote(person_id)
     stmt += " ORDER BY person.person_id"
     dbopenset stmt
     while dbrow()
          print dbstring("person_id") + " " + dbstring("name") + " " + dbstring("phone") + " " + dbstring("type")
     end while
     dbcloseset
end subroutine
```

Exercises:

Word Search

y	p	z	t	c	e	l	e	s	o	x	x	d
e	l	i	b	a	m	l	n	a	x	x	t	b
t	q	x	h	o	o	e	t	g	n	e	d	i
a	s	t	p	s	t	l	n	e	s	t	f	n
e	e	a	e	a	n	i	f	n	t	t	s	t
r	q	t	d	s	r	o	e	b	m	a	d	r
c	n	p	u	t	e	p	i	n	d	b	m	e
i	u	e	s	c	o	s	m	t	c	l	u	s
d	q	b	p	b	e	u	o	l	a	e	y	n
b	d	u	d	o	l	x	o	l	z	l	f	i
r	m	o	e	o	b	s	e	p	c	w	e	m
o	x	h	c	r	e	d	t	b	o	b	y	r
w	c	g	h	t	y	j	c	r	d	s	d	m

	column, create, dbclose, dbcloseset, dbexecute, dbfloat, dbint, dbopen, dbopenset, dbrow, dbstring, insert, query, relationship, row, select, sql, table, update

Problems	1. Take the "Big Program" from this chapter and modify it to create an application to keep track of a student's grades for several classes. You will need the following menu options to allow the user to: • Enter a class code, assignment name, possible points, score on an assignment and store this information into a database table. • Create a way for the student to see all of the grades for a single class after they enter the class code. • Create an option to see a list of all classes with total points possible, total points scored, and percentage of scored vs. possible.