Chapter 17

How to work with a database

Objectives

Applied

1. Use SQLite Manager to test SQL statements against a SQLite database.
2. Develop Python programs that use SQLite databases to store the data of the programs.

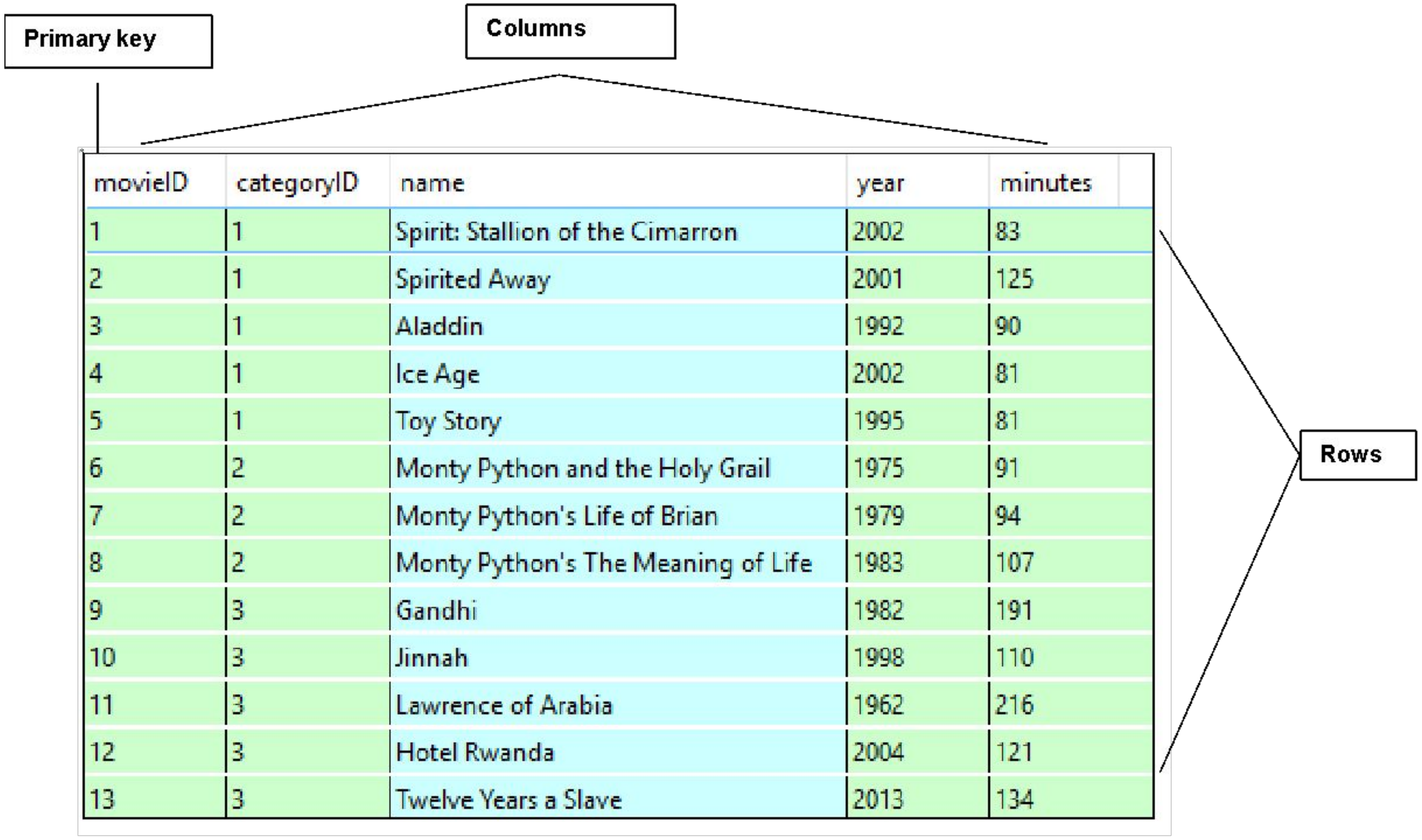
Knowledge

1. Describe the organization of a relational database in terms of tables, rows, columns, primary keys, and foreign keys.
2. Describe a one-to-many relationship between two tables.
3. Describe the way the columns in a table are defined in terms of data types, null values, default values, primary keys, and foreign keys.

Objectives (cont.)

4. Describe the use of these SQL statements: SELECT, INSERT, UPDATE, and DELETE.
5. Describe the use of these clauses in SQL statements: FROM, WHERE, ORDER BY, and JOIN.
6. Describe a result set.
7. Describe the use of these methods of a cursor object: execute(), fetchone(), and fetchall().
8. Describe the use of the commit() method of a connection object.
9. In general terms, explain how to handle database exceptions.

The Movie table

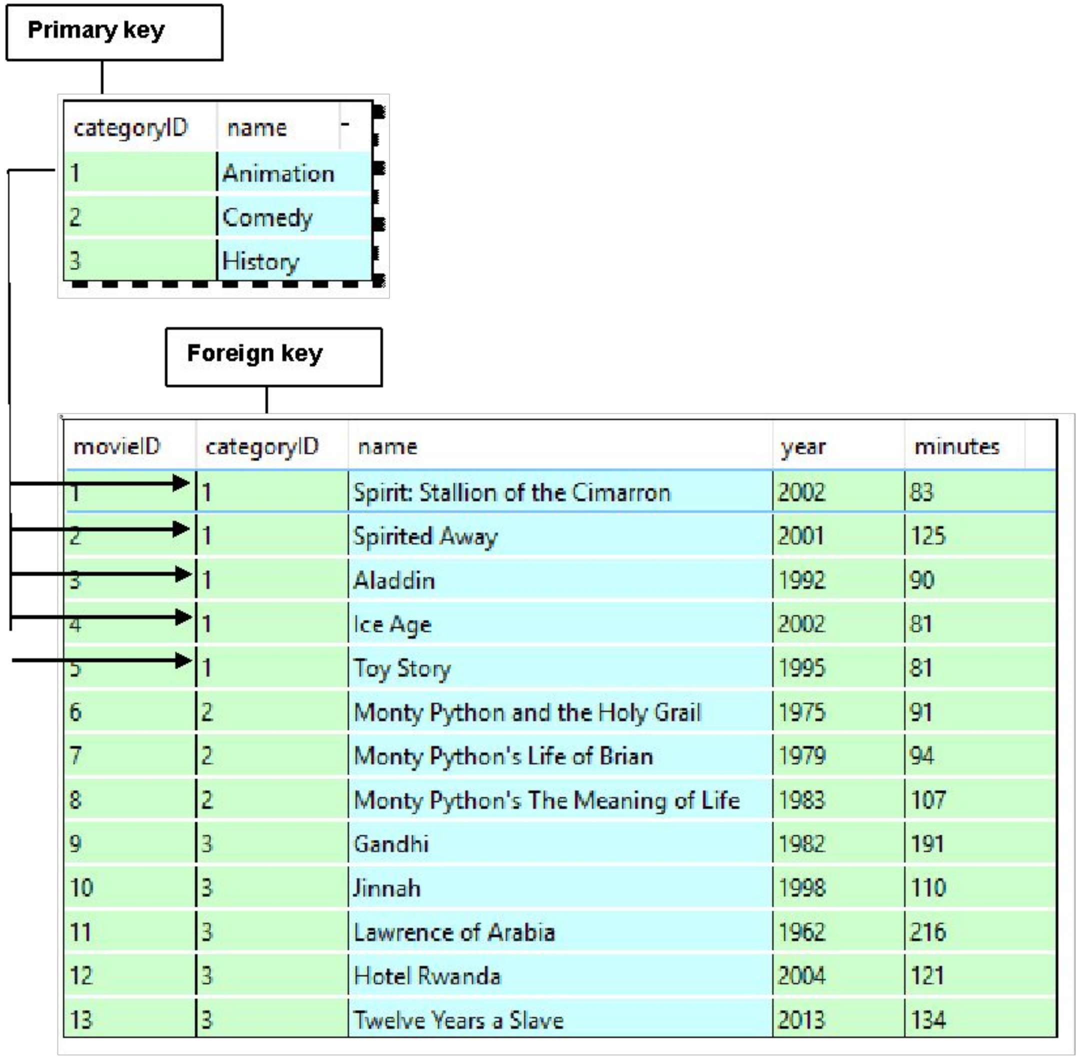


movieID	categoryID	name	year	minutes
1	1	Spirit: Stallion of the Cimarron	2002	83
2	1	Spirited Away	2001	125
3	1	Aladdin	1992	90
4	1	Ice Age	2002	81
5	1	Toy Story	1995	81
6	2	Monty Python and the Holy Grail	1975	91
7	2	Monty Python's Life of Brian	1979	94
8	2	Monty Python's The Meaning of Life	1983	107
9	3	Gandhi	1982	191
10	3	Jinnah	1998	110
11	3	Lawrence of Arabia	1962	216
12	3	Hotel Rwanda	2004	121
13	3	Twelve Years a Slave	2013	134

Concepts

- A *relational database* consists of *tables*. Tables consist of *rows* and *columns*, which can also be referred to as *records* and *fields*.
- A table is typically modeled after a real-world entity, such as a product or customer, but it can also be modeled after an abstract concept, such as the data for a game.
- A column represents an attribute of the entity, such as a movie's name.
- A row contains a set of values for one instance of the entity, such as one movie.
- Most tables have a *primary key* that uniquely identifies each row in the table.
- The primary key is usually a single column, but it can also consist of two or more columns.

The relationship between two tables



categoryID	name
1	Animation
2	Comedy
3	History

movieID	categoryID	name	year	minutes
1	1	Spirit: Stallion of the Cimarron	2002	83
2	1	Spirited Away	2001	125
3	1	Aladdin	1992	90
4	1	Ice Age	2002	81
5	1	Toy Story	1995	81
6	2	Monty Python and the Holy Grail	1975	91
7	2	Monty Python's Life of Brian	1979	94
8	2	Monty Python's The Meaning of Life	1983	107
9	3	Gandhi	1982	191
10	3	Jinnah	1998	110
11	3	Lawrence of Arabia	1962	216
12	3	Hotel Rwanda	2004	121
13	3	Twelve Years a Slave	2013	134

Concepts

- The tables in a relational database are related to each other through their key columns. For example, the Category and Movie tables in this figure use the categoryID column to create the relationship between the two tables.
- The categoryID column in the Movie table is called a *foreign key* because it identifies a related row in the Category table. A table may contain one or more foreign keys.
- When you define a foreign key, you can't add rows to the table with the foreign key unless there's a matching primary key in the related table.
- The relationships between the tables in a database correspond to the relationships between the entities they represent. The most common type of relationship is a *one-to-many relationship* as illustrated by the Category and Movie tables.
- A table can also have a *one-to-one relationship* or a *many-to-many relationship* with another table.

The columns of the Category table

Name	Data Type	Not Null?	Default Value	Primary Key?	Foreign Key?
`categoryID`	`INTEGER`	`Y`	`NULL`	`N`	`N`
`name`	`TEXT`	`Y`	`NULL`	`N`	`N`

The columns of the Movie table

Name	Data Type	Not Null?	Default Value	Primary Key?	Foreign Key?
movieID	INTEGER	Y	NULL	Y	N
categoryID	INTEGER	Y	NULL	N	Y
name	TEXT	Y	NULL	N	N
year	INTEGER	Y	NULL	N	N
minutes	INTEGER	Y	NULL	N	N

Common SQLite data types

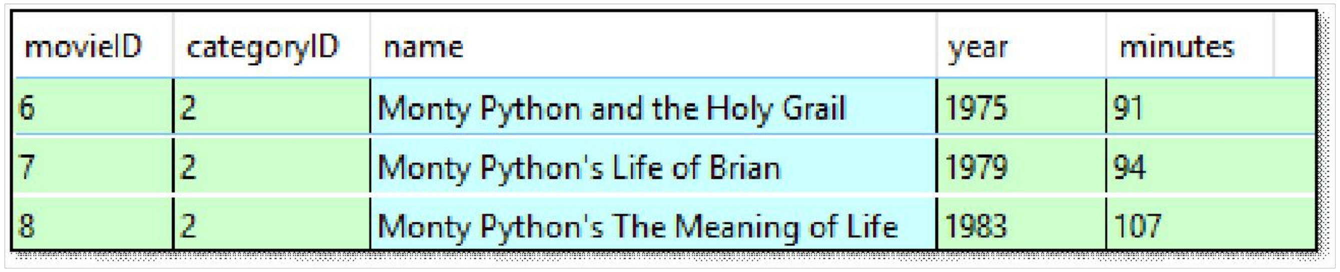
```
TEXT
INTEGER
REAL
BLOB
```

The syntax for a SELECT statement that gets all columns

```
SELECT *
FROM table
[WHERE selection-criteria]
[ORDER BY col-1 [ASC|DESC] [, col-2 [ASC|DESC] ...]]
```

A SELECT statement that gets all columns

```
SELECT * FROM Movie
WHERE categoryID = 2
```



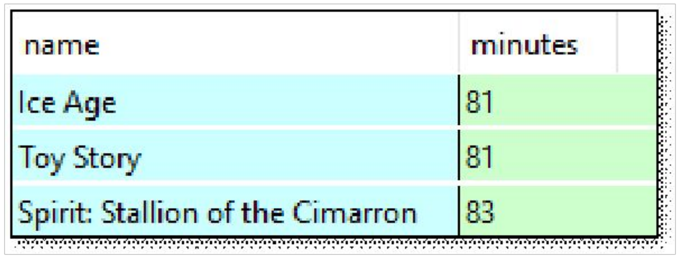
movieID	categoryID	name	year	minutes
6	2	Monty Python and the Holy Grail	1975	91
7	2	Monty Python's Life of Brian	1979	94
8	2	Monty Python's The Meaning of Life	1983	107

The syntax for a SELECT statement that gets selected columns

```
SELECT column-1[, column-2] ...
FROM table
[WHERE selection-criteria]
[ORDER BY column-1 [ASC|DESC][, column-2 [ASC|DESC] ...]]
```

A SELECT statement that gets selected columns and rows

```
SELECT name, minutes
FROM Movie
WHERE minutes < 90
ORDER BY minutes ASC
```



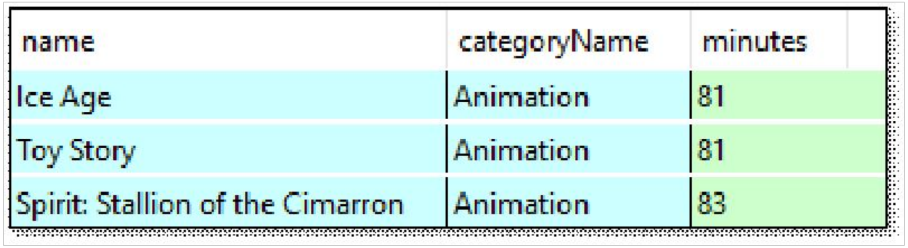
name	minutes
Ice Age	81
Toy Story	81
Spirit: Stallion of the Cimarron	83

The syntax for a SELECT statement that joins two tables

```
SELECT col-1 [AS alias-1] [[, col-2] [AS alias-2]]...
FROM table-1
    [INNER ]JOIN table-2 ON table-1.col-1 = table-2.col-2
```

A statement that gets data from two related tables

```
SELECT Movie.name, Category.name AS categoryName, minutes
FROM Movie
    JOIN Category
        ON Category.categoryID = Movie.categoryID
WHERE minutes < 90
ORDER BY minutes ASC
```



name	categoryName	minutes
Ice Age	Animation	81
Toy Story	Animation	81
Spirit: Stallion of the Cimarron	Animation	83

The syntax for the INSERT statement

```
INSERT INTO table-name [(column-list)]
VALUES (value-list)
```

A statement that uses a column list to add one row

```
INSERT INTO Movie (name, year, minutes, categoryID)
VALUES ('Juno', 2007, 96, 2)
```

A statement that doesn’t use a column list to add one row

```
INSERT INTO Movie
VALUES (14, 2, 'Juno', 2007, 96)
```

The syntax for the UPDATE statement

```
UPDATE table-name
SET expression-1 [, expression-2] ...
WHERE selection-criteria
```

A statement that updates a column in one row

```
UPDATE Movie
SET minutes = 84
WHERE movieID = 4
```

The syntax for the DELETE statement

```
DELETE FROM table-name
WHERE selection-criteria
```

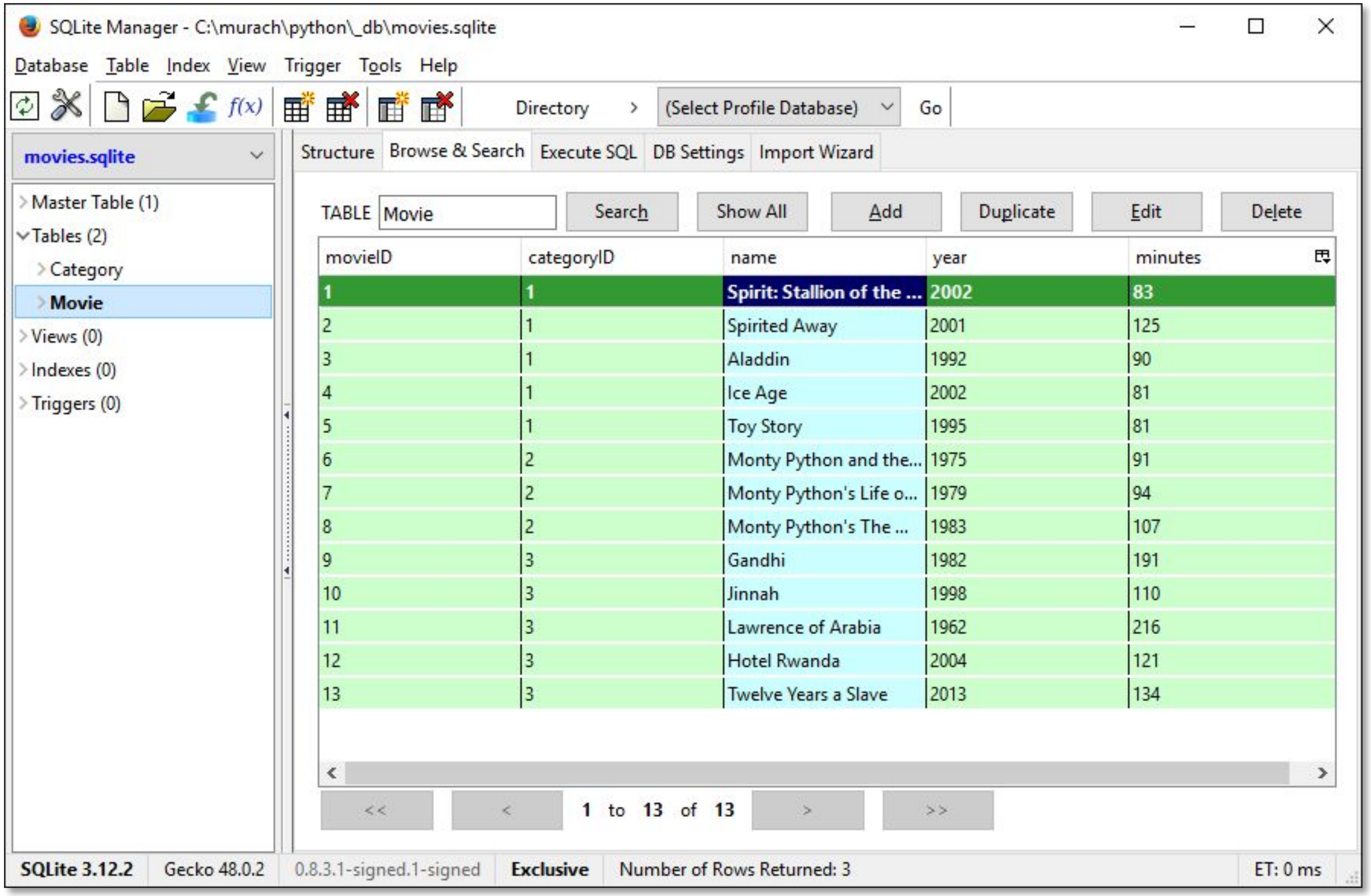
A statement that deletes one row from a table

```
DELETE FROM Movie
WHERE movieID = 14
```

A statement that deletes multiple rows from a table

```
DELETE FROM Movie
WHERE year = 1979
```

SQLite Manager's Browse & Search tab



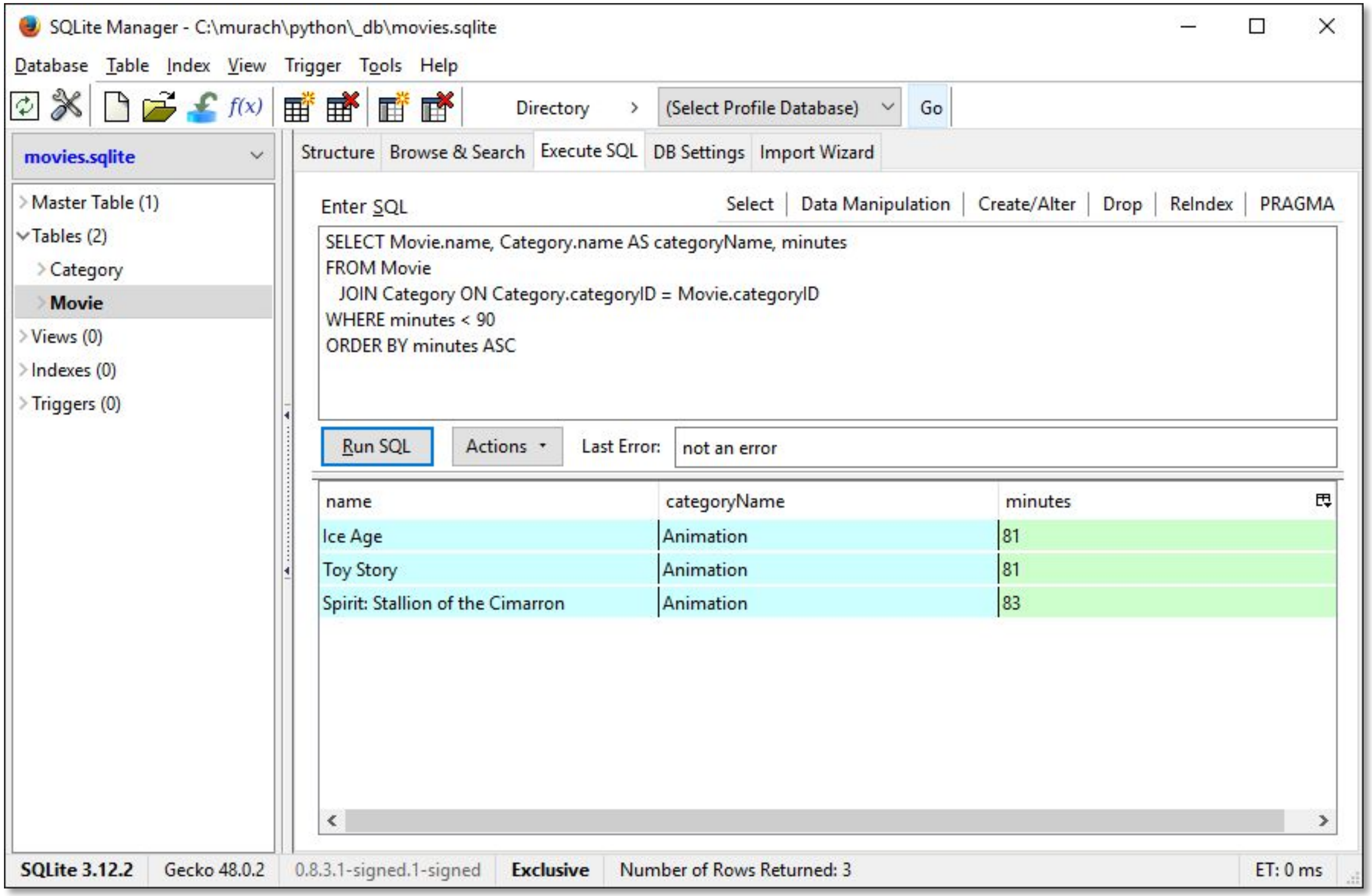
movieID	categoryID	name	year	minutes
1	1	Spirit: Stallion of the ...	2002	83
2	1	Spirited Away	2001	125
3	1	Aladdin	1992	90
4	1	Ice Age	2002	81
5	1	Toy Story	1995	81
6	2	Monty Python and the...	1975	91
7	2	Monty Python's Life o...	1979	94
8	2	Monty Python's The ...	1983	107
9	3	Gandhi	1982	191
10	3	Jinnah	1998	110
11	3	Lawrence of Arabia	1962	216
12	3	Hotel Rwanda	2004	121
13	3	Twelve Years a Slave	2013	134

SQLite 3.12.2 | Gecko 48.0.2 | 0.8.3.1-signed.1-signed | Exclusive | Number of Rows Returned: 3 | ET: 0 ms

How to connect to a SQLite database and view a table

1. Start Firefox.
2. Tools→SQLite Manager to start SQLite Manager.
3. Select Database→Connect Database and navigate to the database file.
4. In the main panel, click the Browse & Search tab.
5. In the left panel, expand the Tables node and click the table you want to view.

The Execute SQL tab


SQLite Manager - C:\murach\python_db\movies.sqlite
Database Table Index View Trigger Tools Help
Directory
(Select Profile Database)
Go
movies.sqlite
Master Table (1)
Tables (2)
Category
Movie
Views (0)
Indexes (0)
Triggers (0)
Structure Browse & Search Execute SQL DB Settings Import Wizard
Enter SQL
Select | Data Manipulation | Create/Alter | Drop | ReIndex | PRAGMA
Run SQL
Actions
SQLite 3.12.2
Gecko 48.0.2
0.8.3.1-signed.1-signed
Exclusive
ET: 0 ms

```
SELECT Movie.name, Category.name AS categoryName, minutes
FROM Movie
  JOIN Category ON Category.categoryID = Movie.categoryID
WHERE minutes < 90
ORDER BY minutes ASC
```

Last Error: not an error

name	categoryName	minutes
Ice Age	Animation	81
Toy Story	Animation	81
Spirit: Stallion of the Cimarron	Animation	83

Number of Rows Returned: 3

How to import the SQLite module

```
import sqlite3
```

The syntax for connecting to a database

```
conn = sqlite3.connect(path_to_database_file)
```

How to connect to a db in the working directory

```
conn = sqlite3.connect("movies.sqlite")
```

How to connect to a db not in the working directory

```
import sys
import os

if sys.platform == "win32":                       # Windows
    DB_FILE = "/murach/python/_db/movies.sqlite"
else:                                             # Mac OS X and Linux
    HOME = os.environ["HOME"]
    DB_FILE = HOME + "/Documents/murach/python/_db/movies.sqlite"
conn = sqlite3.connect(DB_FILE)
```

How to close a connection object

```
if conn:
    conn.close()
```

The cursor() method of the connection object

```
cursor()
```

The execute() method of the cursor object

```
execute(sql [params_tuple])
```

How to get a cursor from the connection object

```
c = conn.cursor()
```

How to execute a SELECT statement…

…that doesn't have parameters

```
query = '''SELECT * FROM Movie'''
c.execute(query)
```

…that has a parameter

```
query = '''SELECT * FROM Movie
           WHERE minutes < ?'''
c.execute(query, (90,))
```

How to automatically close the cursor object

The code for importing the closing() function

```
from contextlib import closing
```

The syntax for automatically closing the cursor object

```
with closing(resource) as name:
    statements
```

How to automatically close the cursor object

```
with closing(conn.cursor()) as c:
    query = '''SELECT * FROM Movie'''
    c.execute(query)
```

The fetchone() and fetchall() methods of the cursor object

```
fetchone()

fetchall()
```

How to use the fetchone() method to get a row

```
with closing(conn.cursor()) as c:
    query = '''SELECT * FROM Movie
               WHERE movieID = ?'''
    c.execute(query, (5,))
    movie = c.fetchone()
```

How to access columns by index

```
print("Name:     " + movie[2])
print("Year:     " + str(movie[3]))
print("Minutes:  " + str(movie[4]))
```

How to access columns by name

How to enable column access by name

```
conn.row_factory = sqlite3.Row      # Row is a constant
```

How to access columns by name

```
print("Name:     " + movie["name"])
print("Year:     " + str(movie["year"]))
print("Minutes:  " + str(movie["minutes"]))
```

How to use the fetchall() method to get all rows

```
with closing(conn.cursor()) as c:
    query = '''SELECT * FROM Movie
               WHERE minutes < ?'''
    c.execute(query, (90,))
    movies = c.fetchall()
```

How to loop through all rows

```
for movie in movies:
    print(movie["name"], "|", movie["year"], "|",
          movie["minutes"])
```

The console

```
Spirit: Stallion of the Cimarron | 2002 | 83
Ice Age | 2002 | 81
Toy Story | 1995 | 81
```

The commit() method of the connection object

```
commit()
```

How to execute an INSERT statement

```
name = "Juno"
year = 2007
minutes = 96
categoryID = 2

with closing(conn.cursor()) as c:
    sql = '''INSERT INTO Movie
                 (name, year, minutes, categoryID)
             VALUES
                 (?, ?, ?, ?)'''
    c.execute(sql, (name, year, minutes, categoryID))
    conn.commit()
```

How to execute an UPDATE statement

```
id = 4
minutes = 84

with closing(conn.cursor()) as c:
    sql = '''UPDATE Movie
             SET minutes = ?
             WHERE movieID = ?'''
    c.execute(sql, (minutes, id))
    conn.commit()
```

How to execute a DELETE statement

```
id = 12

with closing(conn.cursor()) as c:
    sql =  '''DELETE FROM Movie
              WHERE movieID = ?'''
    c.execute(query, (id,))
    conn.commit()
```

Code that tests the database

```
# import the sqlite module and closing function
import sqlite3
from contextlib import closing

# connect to the database and set the row factory
conn = sqlite3.connect("movies.sqlite")
conn.row_factory = sqlite3.Row

# execute a SELECT statement - with exception handling
try:
    with closing(conn.cursor()) as c:
        query = '''SELECT * FROM Movie
                   WHERE minutes < ?'''
        c.execute(query, (90,))
        movies = c.fetchall()
except sqlite3.OperationalError as e:
    print("Error reading database -", e)
    movies = None
```

Code that tests the database (cont.)

```
# display the results
if movies != None:
    for movie in movies:
        print(movie["name"], "|", movie["year"], "|",
              movie["minutes"])
    print()

# execute an INSERT statement
name = "A Fish Called Wanda"
year = 1988
minutes = 108
categoryID = 1
with closing(conn.cursor()) as c:
    sql = '''INSERT INTO Movie
               (name, year, minutes, categoryID)
              VALUES
              (?, ?, ?, ?)'''
    c.execute(sql, (name, year, minutes, categoryID))
    conn.commit()
print(name, "inserted.")
```

Code that tests the database (cont.)

```
# execute a DELETE statement
with closing(conn.cursor()) as c:
    sql =  '''DELETE FROM Movie
              WHERE name = ?'''
    c.execute(sql, (name,))
    conn.commit()
print(name, "deleted.")
```

The console

```
Spirit: Stallion of the Cimarron | 2002 | 83
Ice Age | 2002 | 81
Toy Story | 1995 | 81

A Fish Called Wanda inserted.
A Fish Called Wanda deleted.
```

The user interface for the Movie List program

```
The Movie List program

COMMAND MENU
cat  - View movies by category
year - View movies by year
add  - Add a movie
del  - Delete a movie
exit - Exit program

CATEGORIES
1. Animation
2. Comedy
3. History

Command: add
Name: The Lion King
Year: 1994
Minutes: 89
Category ID: 1
The Lion King was added to database.
```

The user interface (cont.)

```
Command:  cat
Category ID:  1

MOVIES - ANIMATION
ID   Name                                 Year    Mins   Category
-----------------------------------------------------------------
1    Spirit: Stallion of the Cimarron     2002    83     Animation
2    Spirited Away                        2001    125    Animation
3    Aladdin                              1992    90     Animation
4    Ice Age                              2002    81     Animation
5    Toy Story                            1995    81     Animation
14   The Lion King                        1994    89     Animation

Command:  exit
Bye!
```

The objects module for the business tier

```
class Movie:
    def __init__(self, id=0, name=None, year=0,
                 minutes=0, category=None):
        self.id = id
        self.name = name
        self.year = year
        self.minutes = minutes
        self.category = category

class Category:
    def __init__(self, id=0, name=None):
        self.id = id
        self.name = name
```

The db module for the database tier

```
import sys
import os
import sqlite3
from contextlib import closing

from objects import Category
from objects import Movie

conn = None

def connect():
    global conn
    if not conn:
        if sys.platform == "win32":
            DB_FILE = "/murach/python/_db/movies.sqlite"
        else:
            HOME = os.environ["HOME"]
            DB_FILE = HOME +
                "/Documents/murach/python/_db/movies.sqlite"
        conn = sqlite3.connect(DB_FILE)
        conn.row_factory = sqlite3.Row
```

The db module for the database tier (cont.)

```
def close():
    if conn:
        conn.close()

def make_category(row):
    return Category(row["categoryID"], row["categoryName"])

def make_movie(row):
    return Movie(row["movieID"], row["name"], row["year"],
row["minutes"],
            make_category(row))

def get_categories():
    query = '''SELECT categoryID, name as categoryName
               FROM Category'''
    with closing(conn.cursor()) as c:
        c.execute(query)
        results = c.fetchall()

    categories = []
    for row in results:
        categories.append(make_category(row))
    return categories
```

The db module for the database tier (cont.)

```
def get_category(category_id):
    query = '''SELECT categoryID, name AS categoryName
               FROM Category WHERE categoryID = ?'''
    with closing(conn.cursor()) as c:
        c.execute(query, (category_id,))
        row = c.fetchone()

    category = make_category(row)
    return category

def get_movies_by_category(category_id):
    query = '''SELECT movieID, Movie.name, year, minutes,
                      Movie.categoryID as categoryID,
                      Category.name as categoryName
               FROM Movie JOIN Category
                      ON Movie.categoryID = Category.categoryID
               WHERE Movie.categoryID = ?'''
    with closing(conn.cursor()) as c:
        c.execute(query, (category_id,))
        results = c.fetchall()
```

The db module for the database tier (cont.)

```
        movies = []
        for row in results:
            movies.append(make_movie(row))
        return movies

    def get_movies_by_year(year):
        query = '''SELECT movieID, Movie.name, year, minutes,
                          Movie.categoryID as categoryID,
                          Category.name as categoryName
                   FROM Movie JOIN Category
                          ON Movie.categoryID = Category.categoryID
                   WHERE year = ?'''
        with closing(conn.cursor()) as c:
            c.execute(query, (year,))
            results = c.fetchall()

        movies = []
        for row in results:
            movies.append(make_movie(row))
        return movies
```

The db module for the database tier (cont.)

```
def add_movie(movie):
    sql = '''INSERT INTO Movie (categoryID, name, year, minutes)
             VALUES (?, ?, ?, ?)'''
    with closing(conn.cursor()) as c:
        c.execute(sql, (movie.category.id, movie.name,
                        movie.year, movie.minutes))
        conn.commit()

def delete_movie(movie_id):
    sql = '''DELETE FROM Movie WHERE movieID = ?'''
    with closing(conn.cursor()) as c:
        c.execute(sql, (movie_id,))
        conn.commit()
```

The ui module for the presentation tier

```
#!/usr/bin/env/python3

import db
from objects import Movie

def display_title():
    print("The Movie List program")
    print()
    display_menu()

def display_menu():
    print("COMMAND MENU")
    print("cat  - View movies by category")
    print("year - View movies by year")
    print("add  - Add a movie")
    print("del  - Delete a movie")
    print("exit - Exit program")
    print()
```

The ui module for the presentation tier (cont.)

```
def display_categories():
    print("CATEGORIES")
    categories = db.get_categories()
    for category in categories:
        print(str(category.id) + ". " + category.name)
    print()

def display_movies(movies, title_term):
    print("MOVIES - " + title_term)
    line_format = "{:3s} {:37s} {:6s} {:5s} {:10s}"
    print(line_format.format("ID", "Name", "Year", "Mins",
                             "Category"))
    print("-" * 64)
    for movie in movies:
        print(line_format.format(str(movie.id), movie.name,
                                 str(movie.year),
                                 str(movie.minutes),
                                 movie.category.name))
    print()
```

The ui module for the presentation tier (cont.)

```
def display_movies_by_category():
    category_id = int(input("Category ID: "))
    category = db.get_category(category_id)
    if category == None:
        print("There is no category with that ID.\n")
    else:
        print()
        movies = db.get_movies_by_category(category_id)
        display_movies(movies, category.name.upper())

def display_movies_by_year():
    year = int(input("Year: "))
    print()
    movies = db.get_movies_by_year(year)
    display_movies(movies, str(year))
```

The ui module for the presentation tier (cont.)

```
def add_movie():
    name        = input("Name: ")
    year        = int(input("Year: "))
    minutes     = int(input("Minutes: "))
    category_id = int(input("Category ID: "))

    category = db.get_category(category_id)
    if category == None:
        print("There is no category with that ID. " +
              "Movie NOT added.\n")
    else:
        movie = Movie(name=name, year=year, minutes=minutes,
                      category=category)
        db.add_movie(movie)
        print(name + " was added to database.\n")

def delete_movie():
    movie_id = int(input("Movie ID: "))
    db.delete_movie(movie_id)
    print("Movie ID " + str(movie_id) +
          " was deleted from database.\n")
```

The ui module for the presentation tier (cont.)

```
def main():
    db.connect()
    display_title()
    display_categories()
    while True:
        command = input("Command:  ")
        if command == "cat":
            display_movies_by_category()
        elif command == "year":
            display_movies_by_year()
        elif command == "add":
            add_movie()
        elif command == "del":
            delete_movie()
        elif command == "exit":
            break
        else:
            print("Not a valid command. Please try again.\n")
            display_menu()
    db.close()
    print("Bye!")

if __name__ == "__main__":
    main()
```