```
/*** CSE 344 Section 01 -- A Tour of SQLite ***/

/* How to start: open a terminal, then type the command:
                sqlite3 database
   where "database" is the name of the database file you want to use.
WARNING: If you don't specify a database file, sqlite3 won't complain,
but your data will be lost!
*/

/* Useful commands for SQLite (not SQL commands!)

.help - lists other . commands
.headers on/off - show/hide column headers in query results
.mode - how to separate the columns in each row/tuple (for better
formatting)
.read 'filename.sql' - read and execute SQL code from the given file
.separator , - changes the separator for importing files to ,
.show - see how we have set our parameters
.import 'file.txt' Table - loads the file 'file.txt' to the table
Table, be careful to set the separator correctly!
.exit - exit from sqlite3
*/

/* The following are all SQL commands. They have to end with a ";" so
that SQLite can read them! */

/*
   Create tables
 */
-- SQLite ignores string length maximums (N in VARCHAR(N))
-- or fixed string lengths (N in CHAR(N)):
--    http://www.sqlite.org/datatype3.html
-- I've left them in so this code will work with other SQL
-- database management systems.
CREATE TABLE Class (
       dept VARCHAR(6),
       number INTEGER,
       title VARCHAR(75),
       PRIMARY KEY (dept, number)
);

-- Older versions of sqlite (including the one in Mac OS 10.6,
unfortunately)
-- do not enforce FOREIGN KEY constraints.  Newer versions are opt-in
-- at both compile time and runtime (with PRAGMA FOREIGN_KEYS = ON):
--   http://www.sqlite.org/foreignkeys.html
CREATE TABLE Teaches (
       username VARCHAR(8),
       dept VARCHAR(6),
       number INTEGER,
       PRIMARY KEY (username, dept, number),
       FOREIGN KEY (username) REFERENCES Instructor(username),
       FOREIGN KEY (dept, number) REFERENCES Class(dept, number)
);

CREATE TABLE Instructor (
       username VARCHAR(8),
       fname VARCHAR(50),
       lname VARCHAR(50),
       started_on CHAR(10),
       PRIMARY KEY (username)
);

/* Delete a table from the database */
DROP TABLE Instructor ;

/*
   Sample data
 */
INSERT INTO Class
       VALUES('CSE', 378, 'Machine Organization and Assembly
Language');
INSERT INTO Class
       VALUES('CSE', 451, 'Introduction to Operating Systems');
INSERT INTO Class
       VALUES('CSE', 461, 'Introduction to Computer Communication
Networks');

INSERT INTO Instructor
       VALUES('zahorjan', 'John', 'Zahorjan', '1985-01-01');
INSERT INTO Instructor
       VALUES('djw', 'David', 'Wetherall', '1999-07-01');
INSERT INTO Instructor
       VALUES('tom', 'Tom', 'Anderson', date('1997-10-01'));
INSERT INTO Instructor
       VALUES('levy', 'Hank', 'Levy', date('1988-04-01'));

INSERT INTO Teaches
       VALUES('zahorjan', 'CSE', 378);
INSERT INTO Teaches
       VALUES('tom', 'CSE', 451);
INSERT INTO Teaches
       VALUES('tom', 'CSE', 461);
INSERT INTO Teaches
       VALUES('zahorjan', 'CSE', 451);
INSERT INTO Teaches
       VALUES('zahorjan', 'CSE', 461);
INSERT INTO Teaches
       VALUES('djw', 'CSE', 461);
INSERT INTO Teaches
```

```
       VALUES('levy', 'CSE', 451);


/*
   Example queries
 */

-- What courses are offered?
SELECT title
FROM Class;

-- What's the first name of the instructor with login 'zahorjan'?
SELECT fname
FROM Instructor
WHERE username = 'zahorjan'
;

-- What 400-level CSE classes are offered?
SELECT *
FROM Class
WHERE dept = 'CSE' AND 400 <= number AND number <= '499'
;
-- If a string is used where a number is expected,
-- SQLite will try to convert the string into the number
-- it represents.  SQLite also does the opposite conversion.

-- What classes have titles starting with Introduction?
SELECT *
FROM Class
WHERE title LIKE 'Introduction%'
;
-- The LIKE operator does simple pattern matching on the left value
-- using the right value as a pattern.  In LIKE patterns, '%' means
-- any number of arbitrary characters

-- If we misspell Introduction as INtroduction, let's catch that
-- by matching any second character:
SELECT *
FROM Class
WHERE title LIKE 'I_troduction%'
;
-- _ in LIKE patterns matches any single character.

/*
   Fun with strings
 */

-- Show the class titles and their lengths.
SELECT title, LENGTH(title)
FROM Class
```

```
;
-- The LENGTH() method computes exactly that.

-- Truncate all class titles to 12 characters.
SELECT dept, number, SUBSTR(title, 1, 12) AS short_title
FROM Class
;
-- You can give aliases to computed columns with AS.
-- SUBSTR(str, start, length) computes the substring of `str'
-- with (optional) `length', starting from index `start'
-- (first character is index 1).

/*
   Date and time representations
 */
-- SQLite does not have a separate data type for dates, times,
-- or combined date and time.  Instead, these are represented
-- as specially formatted strings; dates are represented as yyyy-mm-dd
-- (see http://www.sqlite.org/lang_datefunc.html for more info).

-- Which instructors started before 1990?
SELECT *
FROM Instructor
WHERE started_on < '1990-01-01'
;

-- Which instructors started before now?
-- (Hopefully, this is all of them!)
SELECT *
FROM Instructor
WHERE started_on < DATE('now');
-- DATE() is a special method that formats a date, described in
-- pseudo-English by the parameter string, in the special SQLite
format.

-- Which instructors started on or after January 1, 15 years ago?
SELECT *
FROM Instructor
WHERE started_on >= DATE('now', 'start of year', '-15 years');
-- You can add extra parameters to DATE(), which describe changes
-- in time from the previous date description.  These modifiers
-- stack from left to right - so we start with today's date, move
-- back to the beginning of this year, then move back 15 years
-- (-15 years).

/*
   Example queries involving joins
 */

-- Who teaches CSE 451?
```

```
SELECT i.fname, i.lname
FROM Class AS c, Teaches AS t, Instructor AS i  -- Can give aliases to
tables
WHERE c.dept = T.dept AND         -- Join conditions
      c.number = T.number AND     --  "
      T.username = I.username AND --  "
      C.number = 451
;

-- What courses does jz teach?
SELECT c.dept, c.number
FROM Class c, Teaches t, Instructor i
WHERE c.dept = t.dept AND
      c.number = t.number AND
      t.username = i.username AND
      i.username = 'zahorjan'
;

-- Semantics of joins:
-- FROM clause takes the Cartesian product of all the named relations.
-- WHERE conditions that relate tuples in two tables implement the
join by
-- filtering the Cartesian product to only those matchings of tuples
that
-- meet the conditions.

-- Which courses do both Hank and John teach?
SELECT c.dept, c.number, c.title
FROM Class c, Teaches t1, Teaches t2, Instructor i1, Instructor i2
WHERE c.dept = t1.dept AND c.dept = t2.dept AND
      c.number = t1.number AND c.number = t2.number AND
      t1.username = i1.username AND
      i1.username = 'levy' AND
      t2.username = i2.username AND
      i2.username = 'zahorjan'
;
-- We can use the same table multiple times in the same query.
-- In fact, in this query, we can't use Teaches or Instructor just
once.
-- Why? Because with just one of both, we'd be asking for tuples
-- where the uid is levy and zahorjan in the same tuple.

-- Which courses do neither Hank nor David teach?
-- wrong --- why?
SELECT c.dept, c.number, c.title
FROM Class c, Teaches t, Instructor i
WHERE c.dept = t.dept AND
      c.number = t.number AND
      t.username = i.username AND
      i.username NOT IN ('levy', 'djw')
```

```
;
-- The above returns two dupes of CSE 451 and 461, because
-- there are tuples in the join where the uid is neither levy or djw,
-- but the class is 451 and 461 -- this comes about from the fact
-- that tom and zahorjan teach those classes.

-- Here's a corrected version that tests that the *class number*
-- is not in the list that Hank and David teach:
-- The query below uses subquerions
-- You will learn about subqueries in a few lectures. No need for hw1
nor hw2:
SELECT *
FROM Class c
WHERE c.dept = 'CSE' AND
      c.number NOT IN (
        SELECT c.number
        FROM Class c, Teaches t, Instructor i
        WHERE c.dept = t.dept AND c.number = t.number AND
              t.username = i.username AND
              i.username IN ('levy', 'djw')
)
;
-- This (correctly) returns only CSE 378.
```