

Database Programming (Part 1)

Copyright © 2026 by
Robert M. Dondero, Ph.D.
Princeton University

Objectives

- We will cover:
 - **Databases (DBs) and database management systems (DBMSs)...**
 - With a focus on **relational** DBs and DBMSs...
 - With a focus on the **SQLite** DBMS...
 - With a focus on **programming** with SQLite

Agenda

- **Relational DBs and DBMSs**
- SQL and SQLite
- The SQLite command-line client

Relational DBs and DBMSs

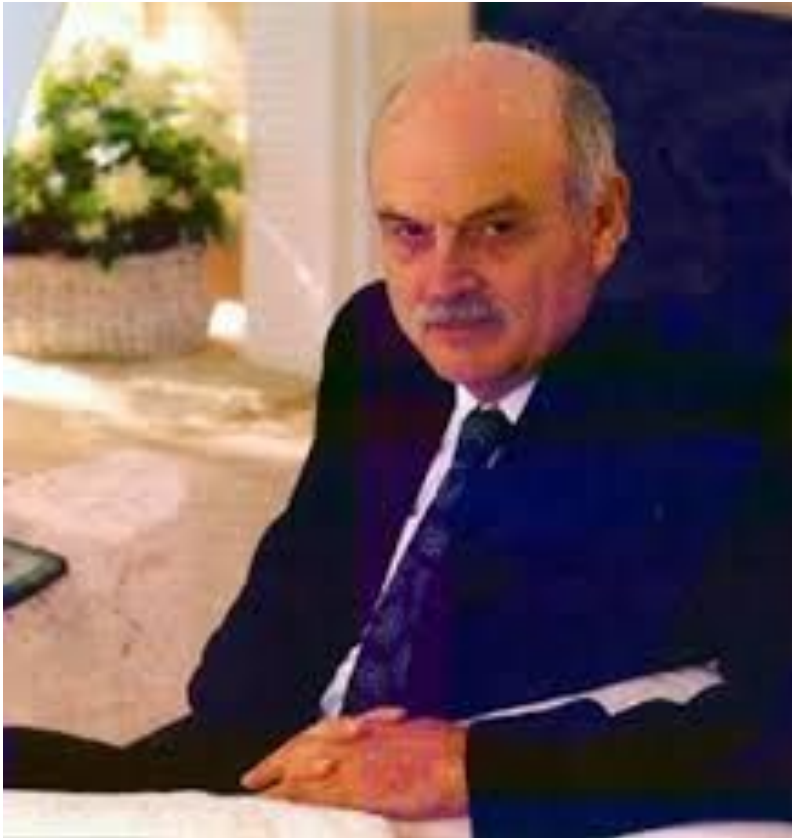
- ***Database (DB)***
 - A structured collection of persistent data
- ***Database management system (DBMS)***
 - Software that maintains DBs
- ***Database administrator (DBA)***
 - A person who administers DBs and DBMSs

Relational DBs and DBMSs

- A good DBMS used by good DBAs can:
 - Reduce redundancy
 - Avoid inconsistencies
 - Facilitate data sharing
 - Enforce standards
 - Apply security restrictions
 - Maintain integrity
 - Balance conflicting requirements
 - Insure safety (backups)

An Introduction to Database Systems, C. J. Date

Relational DBs and DBMSs



Edgar Codd

Relational DBs and DBMSs

Relational DB structure:

Formally	Informally
Relations	Tables
Tuples	Rows
Attributes	Fields

Relational DBs and DBMSs

BOOKS

isbn	title	quantity
123	The Practice of Programming	500
234	The C Programming Language	800
345	Algorithms in C	650

ORDERS

isbn	custid	quantity
123	222	20
345	222	100
123	111	30

CUSTOMERS

custid	custname	street	zipcode
111	Princeton	114 Nassau St	08540
222	Harvard	1256 Mass Ave	02138
333	MIT	292 Main St	02142

AUTHORS

isbn	author
123	Kernighan
123	Pike
234	Kernighan
234	Ritchie
345	Sedgewick

ZIPCODES

zipcode	city	state
08540	Princeton	NJ
02138	Cambridge	MA
02142	Cambridge	MA

Agenda

- Relational DBs and DBMSs
- **SQL and SQLite**
- The SQLite command-line client

SQL and SQLite

SQL



Donald
Chamberlin



Raymond
Boyce

SQL and SQLite

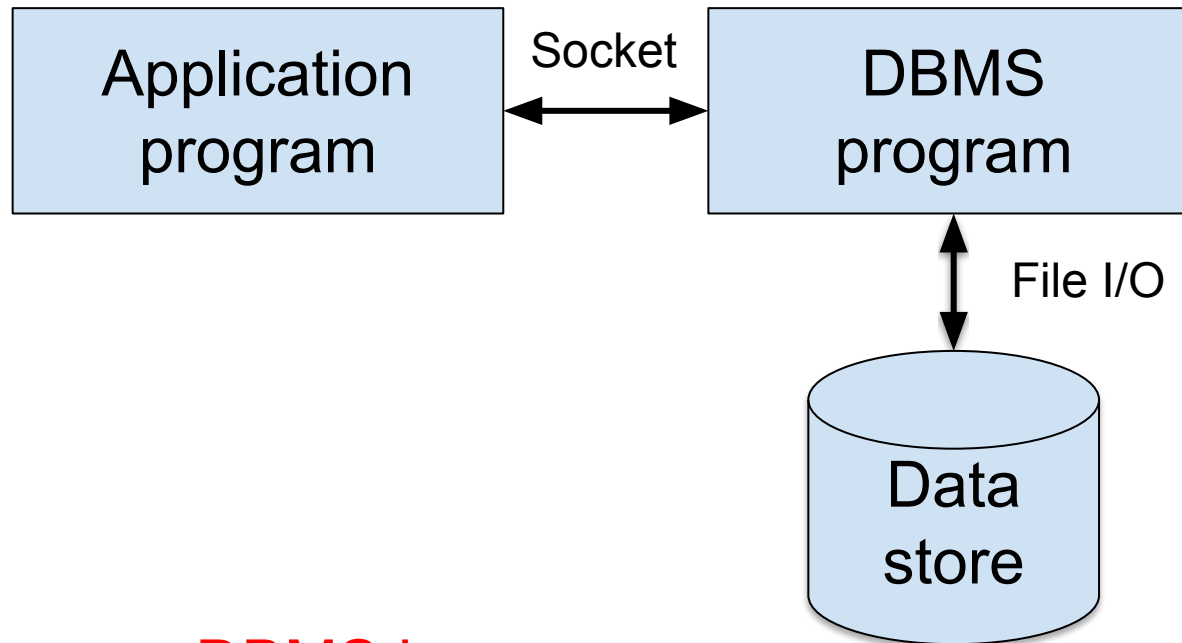
SQLite



D. Richard Hipp

SQL and SQLite

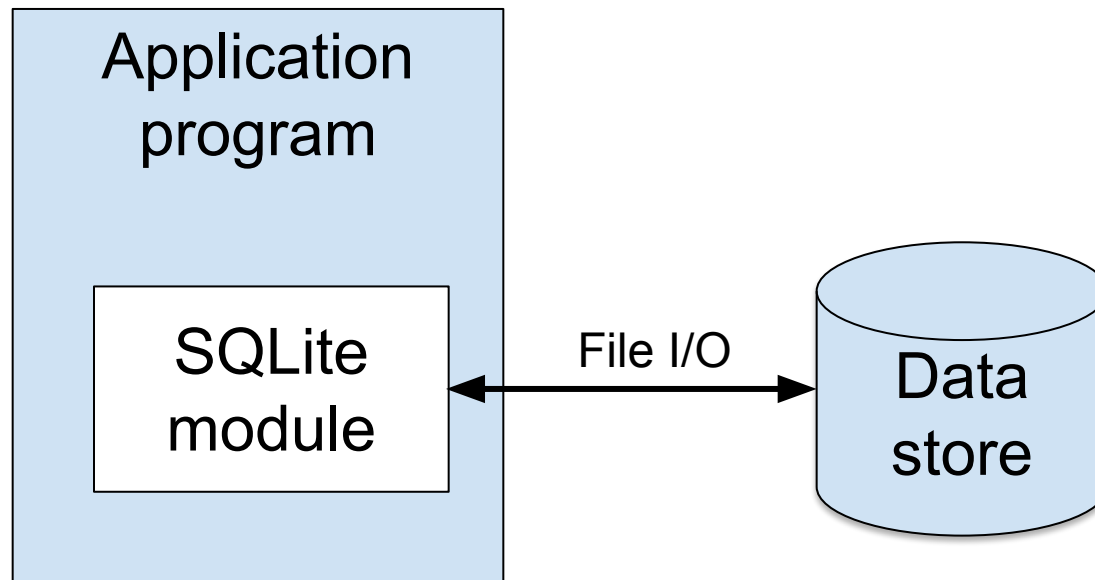
Architecture when using a typical DBMS:



DBMS is a program

SQL and SQLite

Architecture when using SQLite:



DBMS is a module

SQL and SQLite

SQLite Data Type	Python Data Type
INTEGER	int
REAL	float
TEXT	str
BLOB	bytes

Python: None

SQLite: NULL

Agenda

- Relational DBs and DBMSs
- SQL and SQLite
- **The SQLite command-line client**

SQLite Client

- **Question:** How does one use SQLite?
- **Answer:** In this course:
 - Via the SQLite command-line client
 - Via programs that you compose

SQLite Client: Getting Started

```
$ sqlite3 bookstore.sqlite  
SQLite version 3.45.1 2024-01-30 16:01:20  
Enter ".help" for usage hints.  
sqlite>
```

SQLite Client: Getting Started

Standard SQL Statements	SQLite Statements
Do not begin with a period	Begin with a period
Contain keywords are case insensitive	Contain keywords are case sensitive
Must end with a semicolon	Need not end with a semicolon

SQLite Client: Getting Started

.help [-all] [*pattern*]

```
sqlite> .help
.help ?-all? ?PATTERN?  Show help text for PATTERN
.quit                  Exit this program
.schema ?PATTERN?       Show the CREATE statements matching PATTERN
.tables ?TABLE?         List names of tables matching LIKE pattern
TABLE
...
sqlite> .help -all
```

...

SQLite Client: Creating Tables

```
CREATE TABLE [IF NOT EXISTS] table  
(column datatype options, ..., options);
```

```
sqlite> PRAGMA foreign_keys = ON;  
sqlite> CREATE TABLE books (  
    isbn TEXT NOT NULL,  
    title TEXT,  
    quantity INTEGER,  
    PRIMARY KEY (isbn));  
sqlite>
```

SQLite Client: Creating Tables

```
sqlite> CREATE TABLE authors (  
    isbn TEXT NOT NULL,  
    author TEXT NOT NULL,  
    PRIMARY KEY (isbn, author),  
    FOREIGN KEY (isbn) REFERENCES books(isbn));  
sqlite>
```

With foreign key enforcement on, the order of table creation matters

SQLite Client: Creating Tables

```
sqlite> CREATE TABLE zipcodes (  
    zipcode TEXT NOT NULL,  
    city TEXT,  
    state TEXT,  
    PRIMARY KEY (zipcode));  
sqlite>
```

SQLite Client: Creating Tables

```
sqlite> CREATE TABLE customers (  
    custid TEXT NOT NULL,  
    custname TEXT,  
    street TEXT,  
    zipcode TEXT,  
    PRIMARY KEY (custid),  
    FOREIGN KEY (zipcode) REFERENCES zipcodes(zipcode));  
sqlite>
```

SQLite Client: Creating Tables

```
sqlite> CREATE TABLE orders (  
    isbn TEXT NOT NULL,  
    custid TEXT NOT NULL,  
    quantity INTEGER,  
    PRIMARY KEY (isbn, custid),  
    FOREIGN KEY (isbn) REFERENCES books(isbn),  
    FOREIGN KEY (custid) REFERENCES customers(custid));  
sqlite>
```

SQLite Client: Creating Tables

.tables

```
sqlite> .tables
authors      books        customers   orders      zipcodes
sqlite>
```

.schema [*table*]

```
sqlite> .schema books
CREATE TABLE books (
                    isbn TEXT NOT NULL,
                    title TEXT,
                    quantity INTEGER,
                    PRIMARY KEY (isbn));
sqlite>
```

SQLite Client: Inserting Rows

```
INSERT INTO table (column, ...) VALUES (expr, ...);
```

```
sqlite> INSERT INTO books (isbn, title, quantity)
        VALUES ('123','The Practice of Programming',500);
sqlite> INSERT INTO books (isbn, title, quantity)
        VALUES ('234','The C Programming Language',800);
sqlite> INSERT INTO books (isbn, title, quantity)
        VALUES ('345','Algorithms in C',650);
sqlite>
```

SQLite Client: Inserting Rows

```
sqlite> INSERT INTO authors (isbn, author)
VALUES ('123','Kernighan');
sqlite> INSERT INTO authors (isbn, author)
VALUES ('123','Pike');
sqlite> INSERT INTO authors (isbn, author)
VALUES ('234','Kernighan');
sqlite> INSERT INTO authors (isbn, author)
VALUES ('234','Ritchie');
sqlite> INSERT INTO authors (isbn, author)
VALUES ('345','Sedgewick');
sqlite>
```

With foreign key enforcement on, the order of row insertion matters

SQLite Client: Inserting Rows

```
sqlite> INSERT INTO zipcodes (zipcode, city, state)
        VALUES ('08540','Princeton', 'NJ');
sqlite> INSERT INTO zipcodes (zipcode, city, state)
        VALUES ('02138','Cambridge', 'MA');
sqlite> INSERT INTO zipcodes (zipcode, city, state)
        VALUES ('02142','Cambridge', 'MA');
sqlite>
```

SQLite Client: Inserting Rows

```
sqlite> INSERT INTO customers (custid, custname, street, zipcode)
VALUES ('111','Princeton', '114 Nassau St', '08540');
sqlite> INSERT INTO customers (custid, custname, street, zipcode)
VALUES ('222','Harvard', '1256 Mass Ave', '02138');
sqlite> INSERT INTO customers (custid, custname, street, zipcode)
VALUES ('333','MIT', '292 Main St', '02142');
sqlite>
```

SQLite Client: Inserting Rows

```
sqlite> INSERT INTO orders (isbn, custid, quantity)
VALUES ('123','222',20);
sqlite> INSERT INTO orders (isbn, custid, quantity)
VALUES ('345','222',100);
sqlite> INSERT INTO orders (isbn, custid, quantity)
VALUES ('123','111',30);
sqlite>
```

SQLite Client: Selecting Rows

```
SELECT expr, ... FROM table, ... [WHERE condition]  
[ORDER BY column [ASC | DESC]];
```

```
sqlite> SELECT * FROM books;  
123|The Practice of Programming|500  
234|The C Programming Language|800  
345|Algorithms in C|650  
sqlite> SELECT isbn, title FROM books;  
123|The Practice of Programming  
234|The C Programming Language  
345|Algorithms in C  
sqlite> SELECT * FROM books ORDER BY quantity DESC;  
234|The C Programming Language|800  
345|Algorithms in C|650  
123|The Practice of Programming|500  
sqlite>
```

Note: The result is a table

SQLite Client: Selecting Rows

WHERE clauses:

```
sqlite> SELECT * FROM books WHERE quantity=650;  
345|Algorithms in C|650  
sqlite> SELECT * FROM books WHERE quantity>=650;  
234|The C Programming Language|800  
345|Algorithms in C|650  
sqlite> SELECT * FROM orders WHERE isbn=123 AND custid=222;  
123|222|20  
sqlite> SELECT * FROM orders WHERE isbn=123 OR custid=222;  
123|222|20  
345|222|100  
123|111|30  
sqlite>
```

SQLite Client: Selecting Rows

Joining tables:

```
sqlite> SELECT * from books, authors;  
123|The Practice of Programming|500|123|Kernighan  
123|The Practice of Programming|500|123|Pike  
123|The Practice of Programming|500|234|Kernighan  
123|The Practice of Programming|500|234|Ritchie  
123|The Practice of Programming|500|345|Sedgewick  
234|The C Programming Language|800|123|Kernighan  
234|The C Programming Language|800|123|Pike  
234|The C Programming Language|800|234|Kernighan  
234|The C Programming Language|800|234|Ritchie  
234|The C Programming Language|800|345|Sedgewick  
345|Algorithms in C|650|123|Kernighan  
345|Algorithms in C|650|123|Pike  
345|Algorithms in C|650|234|Kernighan  
345|Algorithms in C|650|234|Ritchie  
345|Algorithms in C|650|345|Sedgewick  
sqlite>
```

SQLite Client: Selecting Rows

More reasonable joining of tables:

```
sqlite> SELECT * from books, authors  
        WHERE books.isbn=authors.isbn;  
123|The Practice of Programming|500|123|Kernighan  
123|The Practice of Programming|500|123|Pike  
234|The C Programming Language|800|234|Kernighan  
234|The C Programming Language|800|234|Ritchie  
345|Algorithms in C|650|345|Sedgewick  
sqlite>
```

SQLite Client: Selecting Rows

Alternative (newer) syntax:

```
sqlite> SELECT * from books INNER JOIN authors  
       WHERE books.isbn=authors.isbn;  
123|The Practice of Programming|500|123|Kernighan  
123|The Practice of Programming|500|123|Pike  
234|The C Programming Language|800|234|Kernighan  
234|The C Programming Language|800|234|Ritchie  
345|Algorithms in C|650|345|Sedgewick  
sqlite>
```

SQLite Client: Selecting Rows

Qualifying fields:

```
sqlite> SELECT title, quantity  
        FROM books, orders  
        WHERE books.isbn=orders.isbn;  
Error: ambiguous column name: quantity  
sqlite> SELECT title, orders.quantity  
        FROM books, orders  
        WHERE books.isbn=orders.isbn;  
The Practice of Programming|20  
The Practice of Programming|30  
Algorithms in C|100  
sqlite>
```

SQLite Client: Selecting Rows

Joining more than 2 tables:

```
sqlite> SELECT custname, title, orders.quantity  
        FROM books, customers, orders  
        WHERE books.isbn=orders.isbn  
        AND orders.custid=customers.custid;  
Harvard|The Practice of Programming|20  
Harvard|Algorithms in C|100  
Princeton|The Practice of Programming|30  
sqlite>
```

SQLite Client: Selecting Rows

Joining tables with “missing rows”:

```
sqlite> SELECT * FROM books, orders  
        WHERE books.isbn=orders.isbn;  
123|The Practice of Programming|500|123|111|30  
123|The Practice of Programming|500|123|222|20  
345|Algorithms in C|650|345|222|100  
sqlite>
```

SQLite Client: Selecting Rows

Joining tables with “missing rows” (cont.):

```
123|The Practice of Programming|500|123|222|20
123|The Practice of Programming|500|345|222|100
123|The Practice of Programming|500|123|111|30
234|The C Programming Language|800|123|222|20
234|The C Programming Language|800|345|222|100
234|The C Programming Language|800|123|111|30
345|Algorithms in C|650|123|222|20
345|Algorithms in C|650|345|222|100
345|Algorithms in C|650|123|111|30
```

No row for the book with isbn 234 is in result table

Beware (Assignment 1):

In reg.sqlite some courses have no professors

SQLite Client: Selecting Rows Adv

The `LIKE` operator and wildcards:

```
sqlite> SELECT * FROM books WHERE title LIKE 'The%';  
123|The Practice of Programming|500  
234|The C Programming Language|800  
sqlite> SELECT * FROM books WHERE title LIKE '%of%';  
123|The Practice of Programming|500  
sqlite> SELECT * FROM books WHERE title LIKE 'T_e%';  
123|The Practice of Programming|500  
234|The C Programming Language|800  
sqlite>
```

`%` matches any 0 or more characters

`_` matches any 1 character

SQLite Client: Selecting Rows Adv

Case (in)sensitivity:

```
sqlite> SELECT * FROM authors WHERE author="Pike";  
123|Pike  
sqlite> SELECT * FROM authors WHERE author="pike";  
sqlite> SELECT * FROM books WHERE title LIKE 't_e%';  
123|The Practice of Programming|500  
234|The C Programming Language|800  
sqlite> PRAGMA case_sensitive_like=ON;  
sqlite> SELECT * FROM books WHERE title LIKE 't_e%';  
sqlite>
```

= is case sensitive

LIKE is case insensitive by default

Aside: Escape Char

C, Java, and Python define backslash as the *escape char*

Within a string literal, the char following the escape char is not a special char

```
"abc\"def"
```

The second double quote char doesn't delimit the string, but instead is an ordinary char within the string

SQL doesn't define an escape char, but...

SQLite Client: Selecting Rows Adv

The ESCAPE clause for the LIKE operator

```
sqlite> SELECT * FROM books WHERE title LIKE 'The%';  
123|The Practice of Programming|500  
234|The C Programming Language|800  
sqlite> SELECT * FROM books WHERE title LIKE 'The\%' ESCAPE '\';  
sqlite>
```

SQLite Client: Changing Rows

UPDATE *table* SET *column1=expr1* [, *column2=expr2* ...]
[**WHERE** *condition*]

```
sqlite> UPDATE books SET quantity=60 WHERE isbn=123;
sqlite> SELECT * from books;
123|The Practice of Programming|60
234|The C Programming Language|800
345|Algorithms in C|650
sqlite> UPDATE books SET quantity=quantity+1 WHERE isbn=123;
sqlite> SELECT * from books;
123|The Practice of Programming|61
234|The C Programming Language|800
345|Algorithms in C|650
sqlite> UPDATE books SET quantity=500 WHERE isbn=123;
sqlite> SELECT * from books;
123|The Practice of Programming|500
234|The C Programming Language|800
345|Algorithms in C|650
sqlite>
```

SQLite Client: Deleting Rows

DELETE FROM *table* [WHERE *condition*];

```
sqlite> INSERT INTO books (isbn, title, quantity)
        VALUES ('456', 'Core Java', 120);
sqlite> SELECT * from books;
123|The Practice of Programming|500
234|The C Programming Language|800
345|Algorithms in C|650
456|Core Java|120
sqlite> DELETE FROM books WHERE isbn=456;
sqlite> SELECT * FROM books;
123|The Practice of Programming|500
234|The C Programming Language|800
345|Algorithms in C|650
sqlite>
```

SQLite Client: Deleting Tables

DROP TABLE [IF EXISTS] *table*

```
sqlite> DROP TABLE orders;  
sqlite> DROP TABLE customers;  
sqlite> DROP TABLE zipcodes;  
sqlite> DROP TABLE authors;  
sqlite> DROP TABLE books;  
sqlite> .tables  
sqlite>
```

With foreign key enforcement on, the order of table deletion matters

SQLite Client: Exiting

`.quit`

```
sqlite> .quit  
$
```

SQLite Client

- **Question:** How does one use SQLite?
- **Answer:** In this course:
 - Via the SQLite command-line client
 - Via programs that you compose...

Lecture Summary

- In this lecture we covered:
 - Relational DBs and DBMSs
 - SQL and SQLite
 - The SQLite command-line client
- See also:
 - **Appendix 1:** SQL Client: Reading & Writing
 - **Appendix 2:** Fancy SQL Joins
 - **Appendix 3:** Changing table structure
 - **Appendix 4:** Indices

Appendix 1:

SQL Client: Reading & Writing

SQLite Client: Reading & Writing

To read SQL statements from a text file:

```
$ cat bookstore.sql
PRAGMA foreign_keys = ON;
...
DROP TABLE IF EXISTS orders;
...
CREATE TABLE books (
    isbn TEXT NOT NULL,
    title TEXT,
    quantity INTEGER,
    PRIMARY KEY (isbn));
...
INSERT INTO books (isbn, title, quantity)
    VALUES ('123','The Practice of Programming',500);
...
$
```

SQLite Client: Reading & Writing

To read SQL statements from a text file (cont.):

```
$ sqlite3 bookstore.sqlite  
sqlite> .read bookstore.sql  
sqlite> .quit  
$
```

SQLite Client: Reading & Writing

To write SQL statements to a text file:

```
$ sqlite3 bookstore.sqlite  
sqlite> .output bookstorebackup.sql  
sqlite> .dump  
sqlite> .quit  
$
```

SQLite Client: Reading & Writing

Resulting file:

```
$ cat bookstorebackup.sql
...
CREATE TABLE books (
    isbn TEXT NOT NULL,
    title TEXT,
    quantity INTEGER,
    PRIMARY KEY (isbn));
INSERT INTO books VALUES('123','The Practice of Programming',500);
INSERT INTO books VALUES('234','The C Programming Language',800);
INSERT INTO books VALUES('345','Algorithms in C',650);
CREATE TABLE authors (
...
$
```

Appendix 2: Fancy SQL Joins

Fancy SQL Joins

Recall:

```
sqlite> SELECT * FROM books;  
123|The Practice of Programming|500  
234|The C Programming Language|800  
345|Algorithms in C|650  
sqlite>
```

```
sqlite> SELECT * FROM orders;  
123|222|20  
345|222|100  
123|111|30  
sqlite>
```

```
sqlite> SELECT * FROM books, orders;  
123|The Practice of Programming|500|123|222|20  
123|The Practice of Programming|500|345|222|100  
123|The Practice of Programming|500|123|111|30  
234|The C Programming Language|800|123|222|20  
234|The C Programming Language|800|345|222|100  
234|The C Programming Language|800|123|111|30  
345|Algorithms in C|650|123|222|20  
345|Algorithms in C|650|345|222|100  
345|Algorithms in C|650|123|111|30  
sqlite>
```

Cartesian
product

Fancy SQL Joins

Ordinary SQL join

```
sqlite> SELECT * FROM books, orders WHERE  
books.isbn=orders.isbn;  
123|The Practice of Programming|500|123|111|30  
123|The Practice of Programming|500|123|222|20  
345|Algorithms in C|650|345|222|100  
sqlite>
```

Conceptually, to compute result table:

Compute Cartesian product of `books` and `orders`

Retain only those rows in which `books.isbn = orders.isbn`

Fancy SQL Joins

Inner join

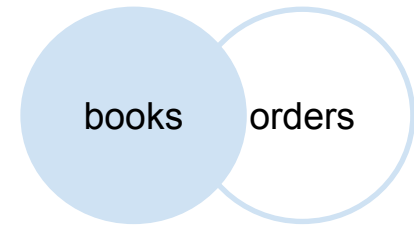
```
sqlite> SELECT * FROM books INNER JOIN orders  
ON books.isbn=orders.isbn;  
123|The Practice of Programming|500|123|111|30  
123|The Practice of Programming|500|123|222|20  
345|Algorithms in C|650|345|222|100  
sqlite>
```

Same as ordinary join

Note: No row for book with `isbn 234` is present

Fancy SQL Joins

Left outer join



```
sqlite> SELECT * FROM books LEFT OUTER JOIN
orders ON books.isbn=orders.isbn;
123|The Practice of Programming|500|123|111|30
123|The Practice of Programming|500|123|222|20
234|The C Programming Language|800|||
345|Algorithms in C|650|345|222|100
sqlite>
```

Conceptually, to compute result table:

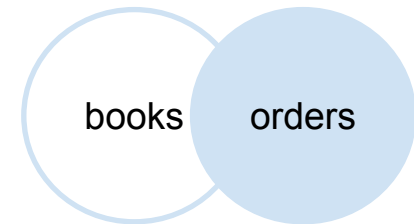
- Compute inner join

- Add each `book` row that is missing, padded with `NULL` fields

Fancy SQL Joins

Right outer join

```
SELECT * from books  
  RIGHT OUTER JOIN orders  
  ON books.isbn = orders.isbn;
```



Conceptually, to compute result table:

- Compute inner join

- Add each `orders` row that is missing, padded with `NULL` fields

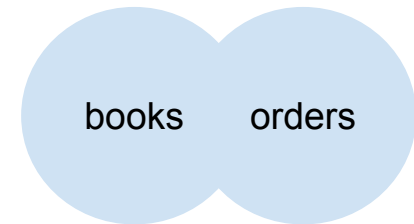
Not supported by SQLite

But could use left outer join with tables switched!

Fancy SQL Joins

Full outer join

```
SELECT * from books
  FULL OUTER JOIN orders
  ON books.isbn = orders.isbn;
```



Conceptually, to compute result table:

- Compute inner join

- Add each `book` row that is missing,
padded with NULL fields

- Add each `orders` row that is missing,
padded with NULL fields

Not supported by SQLite

Fancy SQL Joins

- Note:
 - COS 333 *assignments* do not require outer joins
 - Your COS 333 *project* probably will not require outer joins
 - But understanding outer joins may help you to understand inner joins more deeply

Appendix 3:

Changing Table Structure

Changing Table Structure

ALTER TABLE *table specification* [, *specification*] ...;

```
sqlite> ALTER TABLE books ADD COLUMN price INTEGER;
sqlite> .schema books
CREATE TABLE books (
    isbn TEXT NOT NULL,
    title TEXT,
    quantity INTEGER,
    price INTEGER,
    PRIMARY KEY (isbn));
sqlite> SELECT * FROM books;
123|The Practice of Programming|500|
234|The C Programming Language|800|
345|Algorithms in C|650|
sqlite>
```

Changing Table Structure

Works in standard SQL, but not in SQLite:

```
sqlite> ALTER TABLE books DELETE COLUMN price;  
Error: near "DELETE": syntax error  
sqlite>
```

Works in SQLite:

```
sqlite> ALTER TABLE books RENAME TO books2;  
sqlite> CREATE TABLE books (  
    isbn TEXT NOT NULL,  
    title TEXT,  
    quantity INTEGER,  
    PRIMARY KEY (isbn));  
sqlite> INSERT INTO books (isbn, title, quantity)  
    SELECT isbn, title, quantity from books2;  
sqlite> DROP TABLE books2;
```

Appendix 4: Indices

Indices

Creating indices:

```
CREATE INDEX index ON table (field);
```

```
sqlite> CREATE INDEX books_index ON books (isbn);  
sqlite> .schema books  
CREATE TABLE books (  
    isbn TEXT NOT NULL,  
    title TEXT,  
    quantity INTEGER,  
    PRIMARY KEY (isbn));  
CREATE INDEX books_index on books (isbn);  
sqlite>
```

Pro: Fast *search* on isbn field of books table

Con: Slow *insert* of rows into books table

Con: Slow *delete* of rows from books table

Indices

Dropping indices:

```
DROP INDEX index;
```

```
sqlite> drop index books_index;  
sqlite> .schema books  
CREATE TABLE books (  
                    isbn TEXT NOT NULL,  
                    title TEXT,  
                    quantity INTEGER,  
                    PRIMARY KEY (isbn));  
sqlite>
```