

# Database Programming (Part 1)

Copyright © 2024 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
  - Fundamentals
  - Selecting data
  - Changing data
  - Reading & writing

# Relational DBs and DBMSs

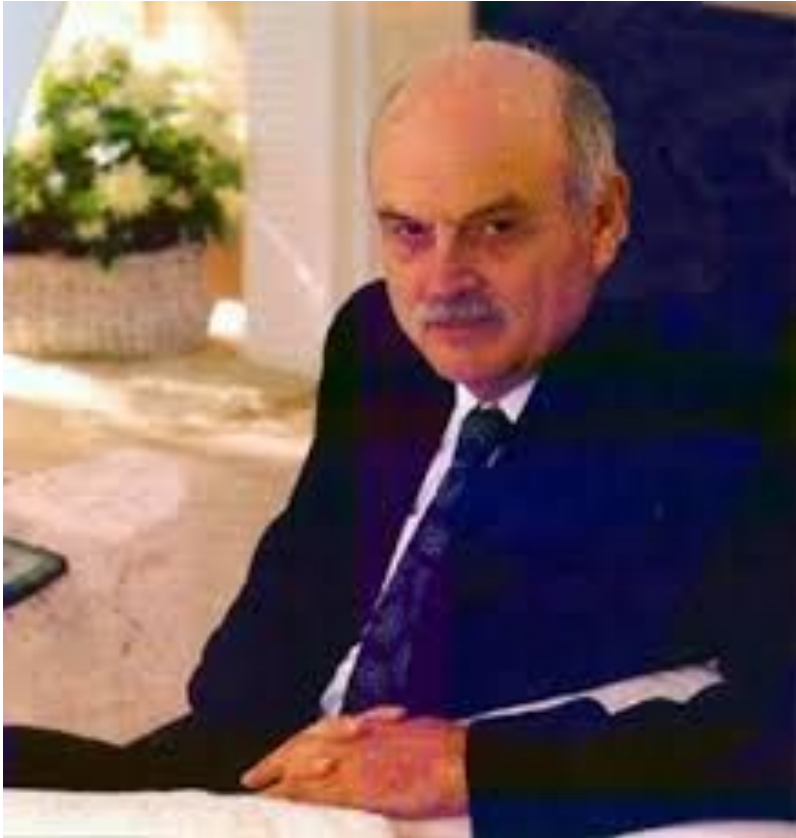
- ***Database (DB)***
  - A structured collection of persistent data
- ***Database management system (DBMS)***
  - Software that maintains DBs
- ***Database administrator (DBA)***
  - A person who administers DBs and DBMSs

# 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:

<b>Formally</b>	<b>Informally</b>
Relations	Tables
Tuples	Rows
Attributes	Fields

# Relational DBs and DBMSs

## BOOKS

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

## AUTHORS

<b>isbn</b>	<b>author</b>
123	Kernighan
123	Pike
234	Kernighan
234	Ritchie
345	Sedgewick

## ORDERS

<b>isbn</b>	<b>custid</b>	<b>quantity</b>
123	222	20
345	222	100
123	111	30

## CUSTOMERS

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

## ZIPCODES

<b>zipcode</b>	<b>city</b>	<b>state</b>
08540	Princeton	NJ
02138	Cambridge	MA
02142	Cambridge	MA



# Agenda

- Relational DBs and DBMSs
- **SQL and SQLite**
- The SQLite command-line client
  - Fundamentals
  - Selecting data
  - Changing data
  - Reading & writing

# SQL and SQLite

## SQL



Donald  
Chamberlin



Raymond  
Boyce

# SQL and SQLite

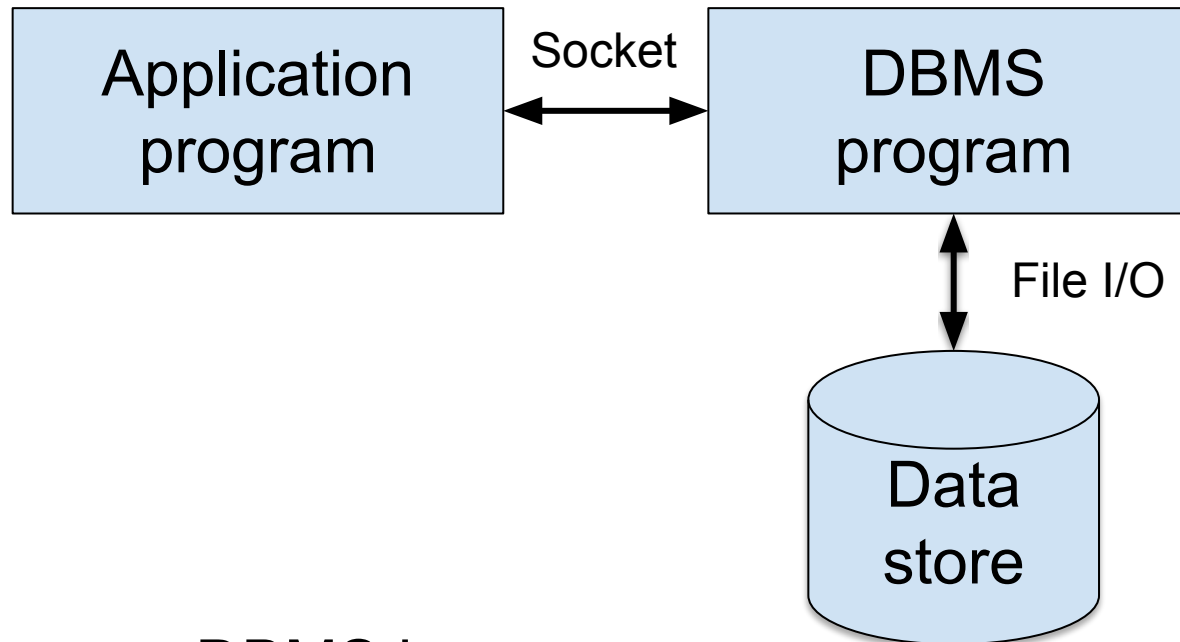
## *SQLite*



D. Richard Hipp

# SQL and SQLite

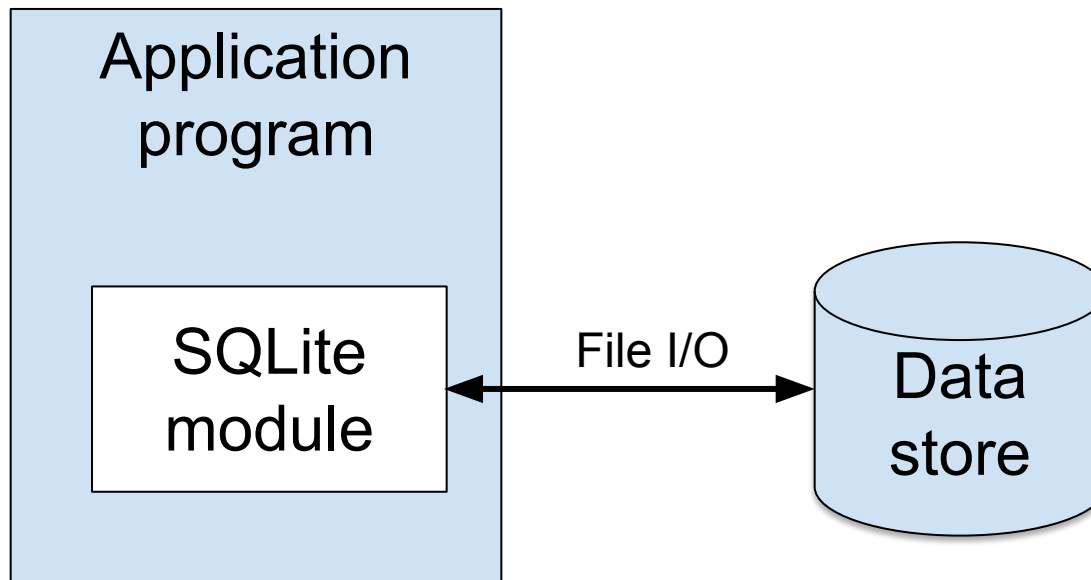
Typical architecture when using a DBMS:



DBMS is a **program**

# SQL and SQLite

Typical architecture when using SQLite:



DBMS is a **module**

# Agenda

- Relational DBs and DBMSs
- SQL and SQLite
- **The SQLite command-line client**
  - Fundamentals
  - Selecting data
  - Changing data
  - Reading & writing

# 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

- The `sqlite3` program

```
$ sqlite3 bookstore.sqlite  
SQLite version 3.31.1 2020-01-27 19:55:54  
Enter ".help" for usage hints.  
sqlite>
```



# SQLite Client

<b>Standard SQL Statements</b>	<b>SQLite Statements</b>
Do not begin with a period	Begin with a period
Keywords are case insensitive	Keywords are case sensitive
Must end with a semicolon	Need not end with a semicolon

# Agenda

- Relational DBs and DBMSs
- SQL and SQLite
- The SQLite command-line client
  - **Fundamentals**
  - Selecting data
  - Changing data
  - Reading & writing

# SQLite Client: Fundamentals

## **.help**

```
sqlite> .help
.dump ?TABLE? ...      Dump the database in an SQL text format
                        If TABLE specified, only dump tables matching
                        LIKE pattern TABLE.

.help                  Show this message
.output ?FILE?         Send output to FILE or stdout
.quit                  Exit this program
.read FILENAME         Execute SQL in FILENAME
.schema ?PATTERN?     Show the CREATE statements matching PATTERN
                        Add --indent for pretty-printing
.tables ?TABLE?       List names of tables
                        If TABLE specified, only list tables matching
                        LIKE pattern TABLE.

...
sqlite>
```

# SQLite Client: Fundamentals

## **.tables**

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

## **.schema [table]**

```
sqlite> .schema  
CREATE TABLE books (isbn TEXT, title TEXT, quantity INTEGER);  
CREATE TABLE authors (isbn TEXT, author TEXT);  
CREATE TABLE customers (custid TEXT, custname TEXT, street TEXT,  
zipcode TEXT);  
CREATE TABLE zipcodes (zipcode TEXT, city TEXT, state TEXT);  
CREATE TABLE orders (isbn TEXT, custid TEXT, quantity INTEGER);  
sqlite> .schema books  
CREATE TABLE books (isbn TEXT, title TEXT, quantity INTEGER);  
sqlite>
```

# SQLite Client: Fundamentals

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

SQLite: NULL

Python: None

# Agenda

- Relational DBs and DBMSs
- SQL and SQLite
- The SQLite command-line client
  - Fundamentals
  - **Selecting data**
  - Changing data
  - Reading & writing

# SQLite Client: Selecting Data

```
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 Data

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 Data

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>
```

# SQLite Client: Selecting Data

Case (in)sensitivity:

```
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>
```

# 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 Data

## 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: Selecting Data

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>
```

Cartesian  
product

# SQLite Client: Selecting Data

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 Data

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 Data

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 Data

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>
```

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 Data

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

```
sqlite> CREATE INDEX books_index ON books (isbn);  
sqlite> .schema books  
CREATE TABLE books  
(isbn TEXT, title TEXT, quantity INTEGER);  
CREATE INDEX books_index ON books (isbn);  
sqlite>
```

# Agenda

- Relational DBs and DBMSs
- SQL and SQLite
- The SQLite command-line client
  - Fundamentals
  - Selecting data
  - **Changing data**
  - Reading & writing

# SQLite Client: Changing Data

**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: Changing Data

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

```
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>
```

# SQLite Client: Changing Data

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

```
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>
```

# Question (lecture04part1)

What is the effect of this SQL statement:

```
sqlite> DELETE FROM books;  
sqlite>
```

Browse to <https://cos333attend.cs.princeton.edu> to answer

# SQLite Client: Changing Data

**DROP TABLE [IF EXISTS] *table***

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



# SQLite Client: Changing Data

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

```
sqlite> CREATE TABLE books (isbn TEXT, title TEXT,  
quantity INTEGER);  
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> SELECT * FROM books;  
123|The Practice of Programming|500  
234|The C Programming Language|800  
345|Algorithms in C|650  
sqlite>
```

# SQLite Client: Changing Data

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

```
sqlite> ALTER TABLE books ADD COLUMN price INTEGER;  
sqlite> .schema books  
CREATE TABLE books  
(isbn TEXT, title TEXT, quantity INTEGER, price INTEGER);  
sqlite> SELECT * FROM books;  
123|The Practice of Programming|500|  
234|The C Programming Language|800|  
345|Algorithms in C|650|  
sqlite>
```

# SQLite Client: Changing Data

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

```
sqlite> ALTER TABLE books RENAME TO books2;  
sqlite> CREATE TABLE books (isbn TEXT, title TEXT, quantity  
INTEGER);  
sqlite> INSERT INTO books (isbn, title, quantity) SELECT isbn,  
title, quantity from books2;  
sqlite> DROP TABLE books2;  
sqlite> .schema books  
CREATE TABLE books  
(isbn TEXT, title TEXT, quantity INTEGER);  
sqlite>
```

# SQLite Client: Changing Data

`.quit`

```
sqlite> .quit  
$
```

# Agenda

- Relational DBs and DBMSs
- SQL and SQLite
- The SQLite command-line client
  - Fundamentals
  - Selecting data
  - Changing data
  - **Reading & writing**

# SQLite Client: Reading & Writing

To read SQL statements from a text file:

```
$ cat bookstore.sql
CREATE TABLE books
(isbn TEXT, title TEXT, quantity INTEGER);

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

CREATE TABLE authors (isbn TEXT, author TEXT);
...
$
```

# 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, title TEXT, quantity INTEGER);
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 (isbn TEXT, author TEXT);
...
$
```

# SQLite Client: Reading & Writing

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

# Summary

- We have covered:
  - Relational DBs and DBMSs
  - SQL and SQLite
  - The SQLite command-line client
- See also:
  - **Appendix 1: Fancy SQL Joins**

# Appendix 1: 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*

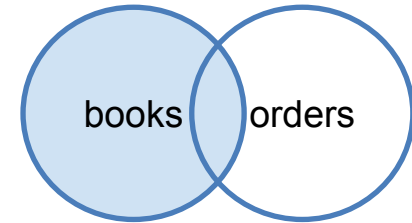
```
sqlite> SELECT * FROM books INNER JOIN orders  
ON books.isbn=orders.isbn;  
123|The Practice of Programming|500|123|111|30  
123|The Practice of Programming|500|123|222|20  
345|Algorithms in C|650|345|222|100  
sqlite>
```

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

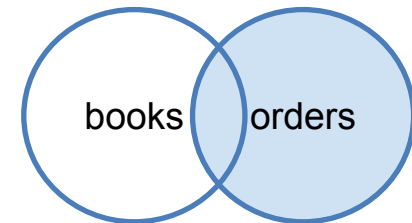
- Add each `book` row that is missing, padded with `NULL` fields



# 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

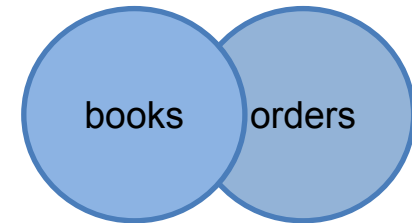
**Not supported by SQLite**

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 better understand inner joins