Database Programming (Part 3)

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 DB transactions: atomicity**
- Relational DB transactions: locking
- Relational DB design

DB Transactions: Atomicity

Customer 222 purchased 1 copy of book 123

Time

```
UPDATE orders SET quantity=quantity+1
WHERE isbn=123 AND custid=222;
```

HW/SW failure

```
UPDATE books SET quantity=quantity-1
WHERE isbn=123;
```

DB Transactions: Atomicity

Preserve consistency with
HW/SW failures

↓ *Requires*

Atomicity

↓ *Implemented by*

Transactions

DB Transactions: Atomicity

```
BEGIN
UPDATE ...
ADD ...
DELETE ...
...
COMMIT
```

DBMS starts a transaction

DBMS stages changes in memory

DBMS writes the staged changes to the DB and ends the transaction

```
BEGIN
UPDATE ...
ADD ...
DELETE ...
...
ROLLBACK
```

DBMS starts a transaction

DBMS stages changes in memory

DBMS discards the staged changes and ends the transaction

DB Transactions: Atomicity

- See **purchase.py**

```
$ python display.py
-------------------------------------------
books
-------------------------------------------
('123', 'The Practice of Programming', 500)
…
-------------------------------------------
orders
-------------------------------------------
('123', '222', 20)
…
$ python purchase.py 123 222
Transaction committed.
```

```
$ python display.py
-------------------------------------------
books
-------------------------------------------
('123', 'The Practice of Programming', 499)
…
-------------------------------------------
orders
-------------------------------------------
('123', '222', 21)
…
$
```

Aside: Isolation Level

```
connect(DATABASE_URL,
    isolation_level=None, uri=True
```

"You can disable the sqlite3 module's implicit transaction management by setting isolation_level to None. This will leave the underlying sqlite3 library operating in autocommit mode. You can then completely control the transaction state by explicitly issuing BEGIN, ROLLBACK, SAVEPOINT, and RELEASE statements in your code."

https://docs.python.org/3/library/sqlite3.html

DB Transactions: Atomicity

- See **recovery.py**

```
$ python display.py
-------------------------------------------
books
-------------------------------------------
('123', 'The Practice of Programming', 500)
…
-------------------------------------------
orders
-------------------------------------------
('123', '222', 20)
…
$
```

DB Transactions: Atomicity

- See **recovery.py** (cont.)

```
$ python recovery.py
Transaction 0 committed.
Transaction 1 committed.
Transaction 2 committed.
Transaction 3 committed.
Transaction 4 committed.
Simulated failure with i = 5
Transaction 5 rolled back.
Transaction 6 committed.
Transaction 7 committed.
Transaction 8 committed.
Transaction 9 committed.
Transaction 10 committed.
Simulated failure with i = 11
Transaction 11 rolled back.
Transaction 12 committed.
Simulated failure with i = 13
Transaction 13 rolled back.
Transaction 14 committed.
Transaction 15 committed.
Transaction 16 committed.
Transaction 17 committed.
Transaction 18 committed.
Transaction 19 committed.
$
```

DB Transactions: Atomicity

- See **recovery.py** (cont.)

```
$ python display.py
-------------------------------------------
books
-------------------------------------------
('123', 'The Practice of Programming', 483)
…
-------------------------------------------
orders
-------------------------------------------
('123', '222', 37)
…
$
```

Agenda

- Relational DB transactions: atomicity
- **Relational DB transactions: locking**
- Relational DB design

DB Transactions: Locking

Without locking:

Time ↓

Process P1	Process P2
BEGIN	
Increment ORDERS (21)	
	Increment ORDERS (21)
Decrement BOOKS (499)	
	Decrement BOOKS (499)
COMMIT	

DB Transactions: Locking

Preserve consistency with concurrent updates

↓ *Requires*

Locking

↓ *Implemented by*

Transactions

DB Transactions: Locking

Time

Process P1	Process P2
BEGIN	
Increment orders (21)	Updates disallowed
Decrement books (499)	
COMMIT	
	Increment orders (22)
	Decrement books (498)

Process P2 updates are postponed or rejected

DB Transactions: Locking

DBMS	Locking Level
PostgreSQL	row
Oracle	row
SQLServer	row
MySQL	row
SQLite	database

DB Transactions: Locking

Transaction locking in SQLite

After P1 does this on some DB	P2 can do this on that DB
BEGIN	**SELECT | UPDATE | ADD | DELETE**
SELECT | UPDATE | ADD | DELETE	**SELECT**
COMMIT | ROLLBACK	**SELECT | UPDATE | ADD | DELETE**

DB Transactions: Locking

Terminal session 1:

```
$ sqlite3 bookstore.sqlite
sqlite> BEGIN;
sqlite> UPDATE books SET quantity = 11111 WHERE isbn = 123;
sqlite>
```

Terminal session 2:

```
$ python purchase.py 123 222
database is locked
$
```

Noticeable delay

DB Transactions: Locking

Terminal session 1 (cont):

```
$ sqlite3 bookstore.sqlite
sqlite> BEGIN;
sqlite> UPDATE books SET quantity = 11111 WHERE isbn = 123;
sqlite> COMMIT;
sqlite>
```

Terminal session 2 (cont):

```
$ python purchase.py 123 222
database is locked
$ python purchase.py 123 222
Transaction committed.
$
```

Transaction Summary

- DBMSs use **transactions** to:
  - Recover from HW/SW errors
    - Transactions implement **atomicity**
  - Handle concurrent updates
    - Transactions implement **locking**

Agenda

- Relational DB transactions: atomicity
- Relational DB transactions: locking
- **Relational DB design**

Relational DB Design

- Relational DB ***normal forms***

Relational DB Design

- Somewhat informally…
- **Def**: A table is in ***first normal form*** iff no cell of a table is a table
  - All modern relational DBMSs enforce first normal form

Relational DB Design: DB1

DB1:

BOOKS

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

AUTHORS

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

ORDERS

isbn	custid	custname	street	city	state	zipcode	quantity
123	222	Harvard	1256 Mass Ave	Cambridge	MA	02138	20
345	222	Harvard	1256 Mass Ave	Cambridge	MA	02138	100
123	111	Princeton	114 Nassau St	Princeton	NJ	08540	30

Relational DB Design: DB1

- Somewhat informally...
- **Def**: The ***primary key*** for a table is the minimal set of columns that uniquely identifies any particular row of that table

Relational DB Design: DB1

Primary keys (boldface) in DB1:

BOOKS

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

AUTHORS

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

ORDERS

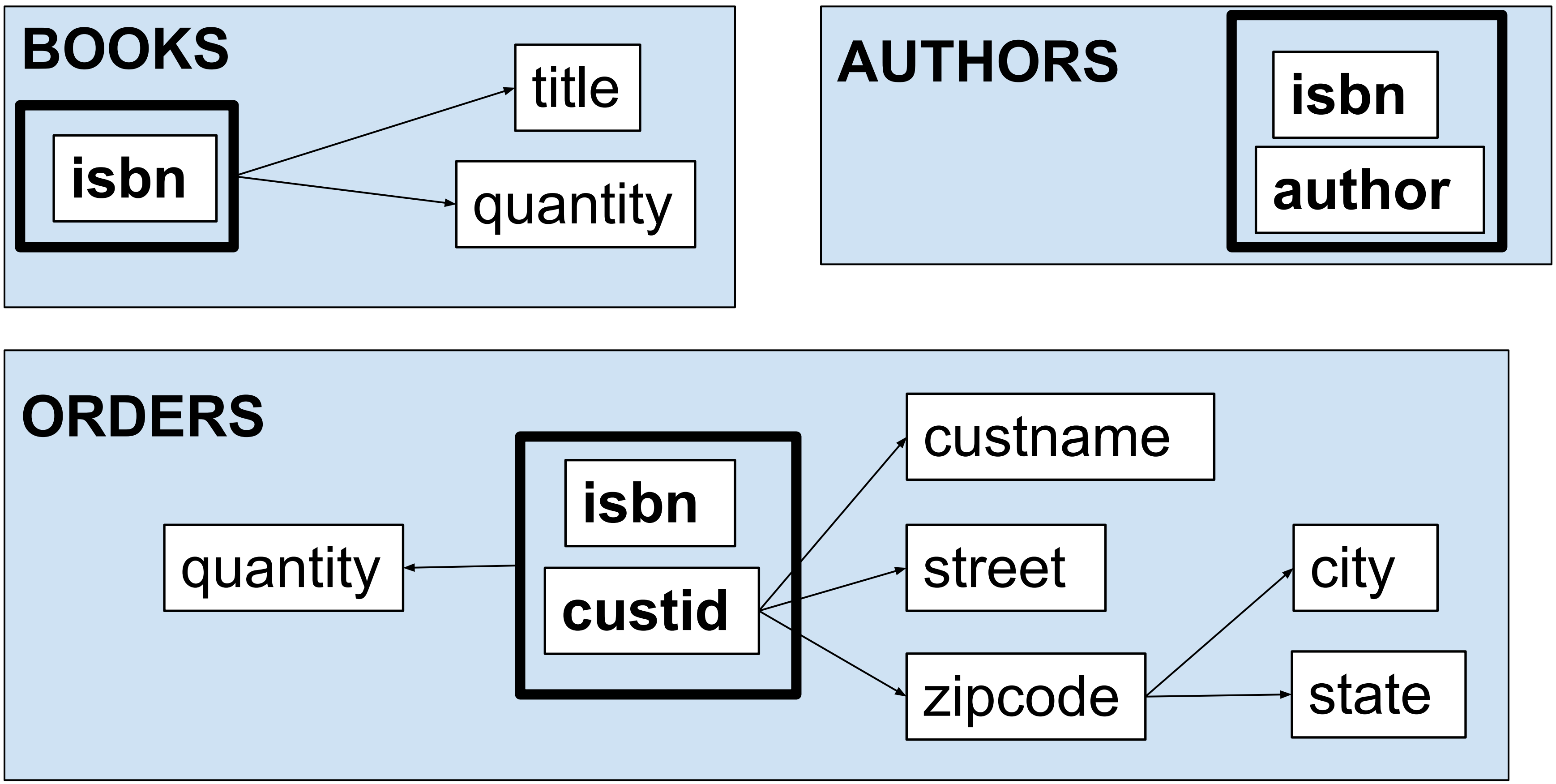
isbn	**custid**	**custname**	**street**	**city**	**state**	**zipcode**	**quantity**
123	**222**	Harvard	1256 Mass Ave	Cambridge	MA	02138	20
345	**222**	Harvard	1256 Mass Ave	Cambridge	MA	02138	100
123	**111**	Princeton	114 Nassau St	Princeton	NJ	08540	30

Relational DB Design: DB1

- **Def**: A column C2 of a table is (functionally) ***dependent*** on a column C1 iff, for each row in the table, the value of C1 determines the value of C2
- In DB1...

Relational DB Design: DB1

Dependencies in DB1:

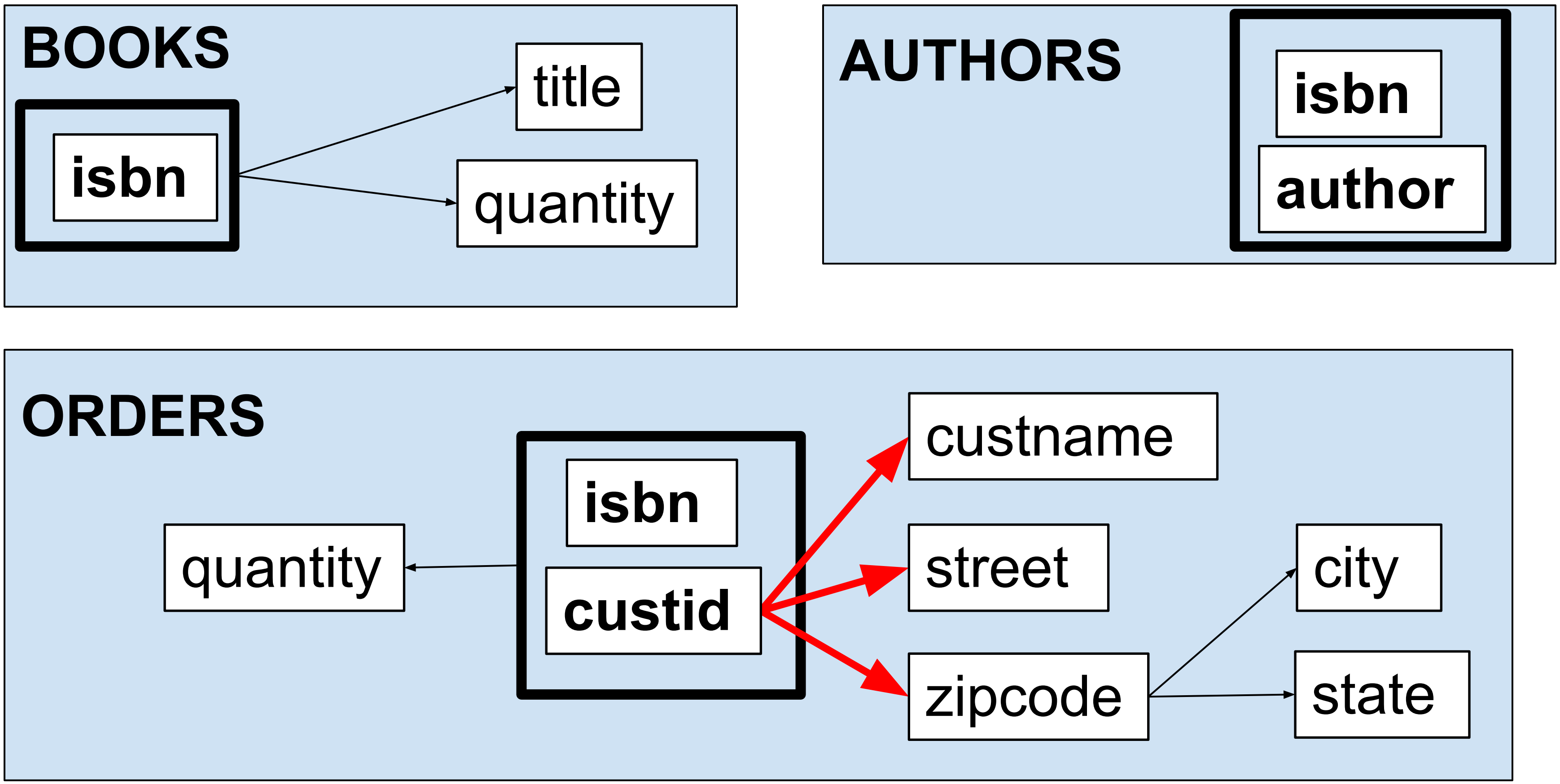


Relational DB Design: DB1

- Somewhat informally…
- A table is in ***second normal form*** iff:
  - It is in first normal form, and
  - Every non-primary-key column is dependent on the **entire** primary key

Relational DB Design: DB1

Dependencies in DB1:



DB1 is not in second normal form

Relational DB Design: DB2

DB2:

BOOKS

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

AUTHORS

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

ORDERS

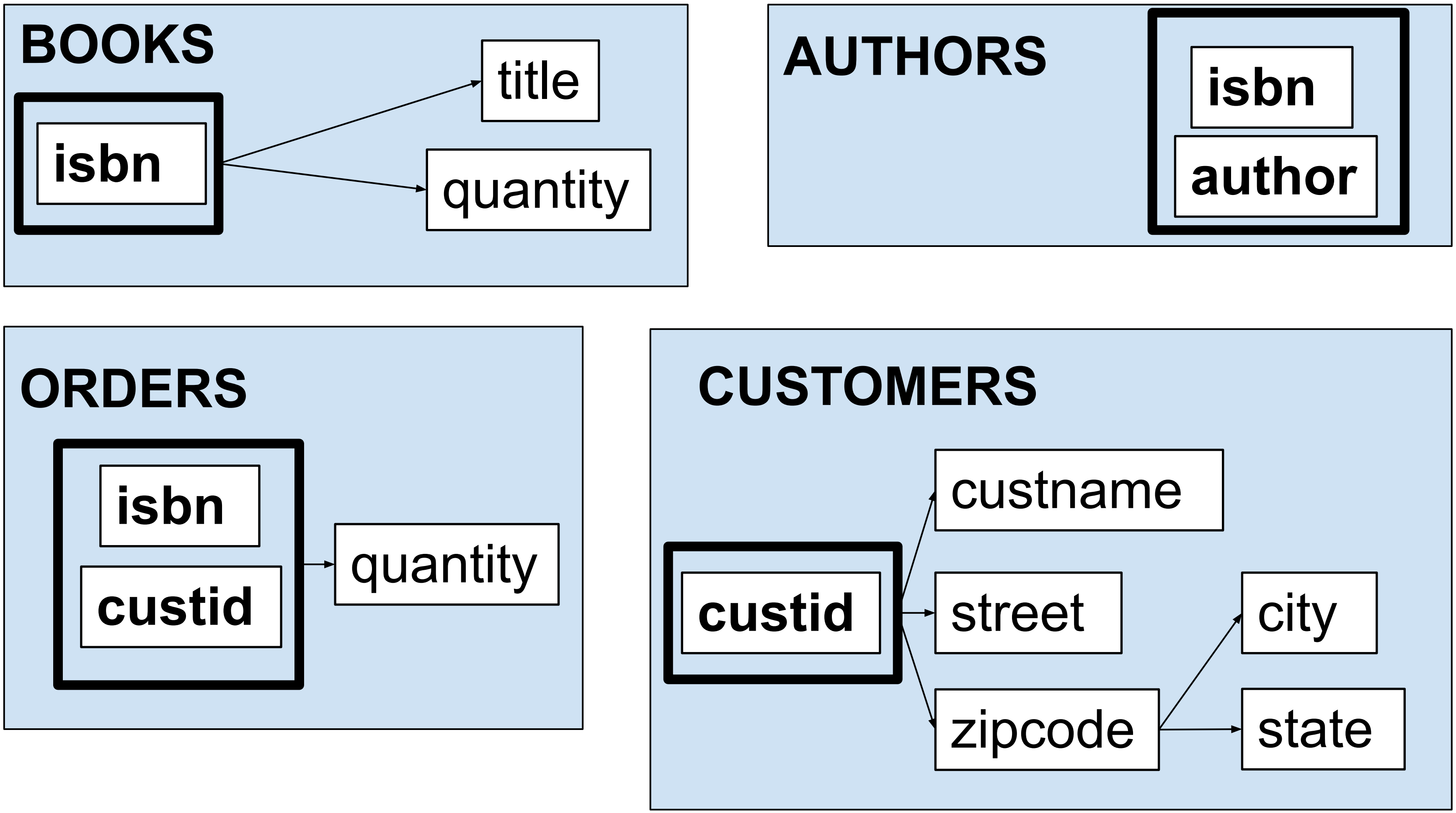
isbn	custid	quantity
123	222	20
345	222	100
123	111	30

CUSTOMERS

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

Relational DB Design: DB2

Dependencies in DB2:



DB2 is in second normal form

Relational DB Design: DB2

DB2:

BOOKS

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

AUTHORS

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

ORDERS

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

Design of DB2 seems wrong

CUSTOMERS

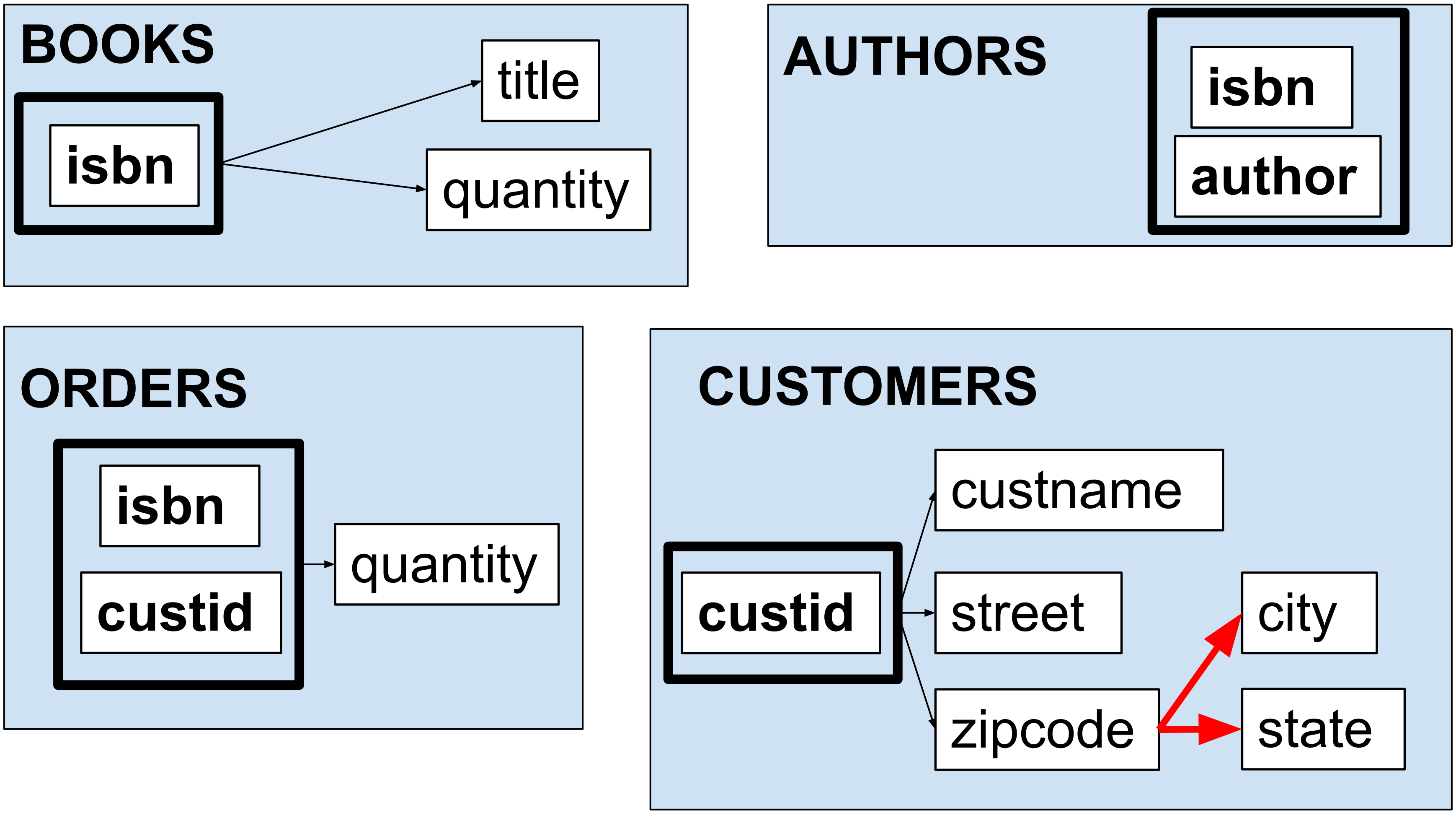
custid	custname	street	city	state	zipcode
111	Princeton	114 Nassau St	Princeton	NJ	08540
222	Harvard	1256 Mass Ave	Cambridge	MA	02138
333	MIT	292 Main St	Cambridge	MA	02142

Relational DB Design: DB2

- Somewhat informally...
- A table is in ***third normal form*** iff:
  - It is in second normal form, and
  - Every non-primary-key column is (non-transitively) **directly** (functionally) dependent on the primary key

Relational DB Design: DB2

Dependencies in DB2:



DB2 is not in third normal form

Relational DB Design: DB3

DB3:

BOOKS

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

AUTHORS

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

ORDERS

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

CUSTOMERS

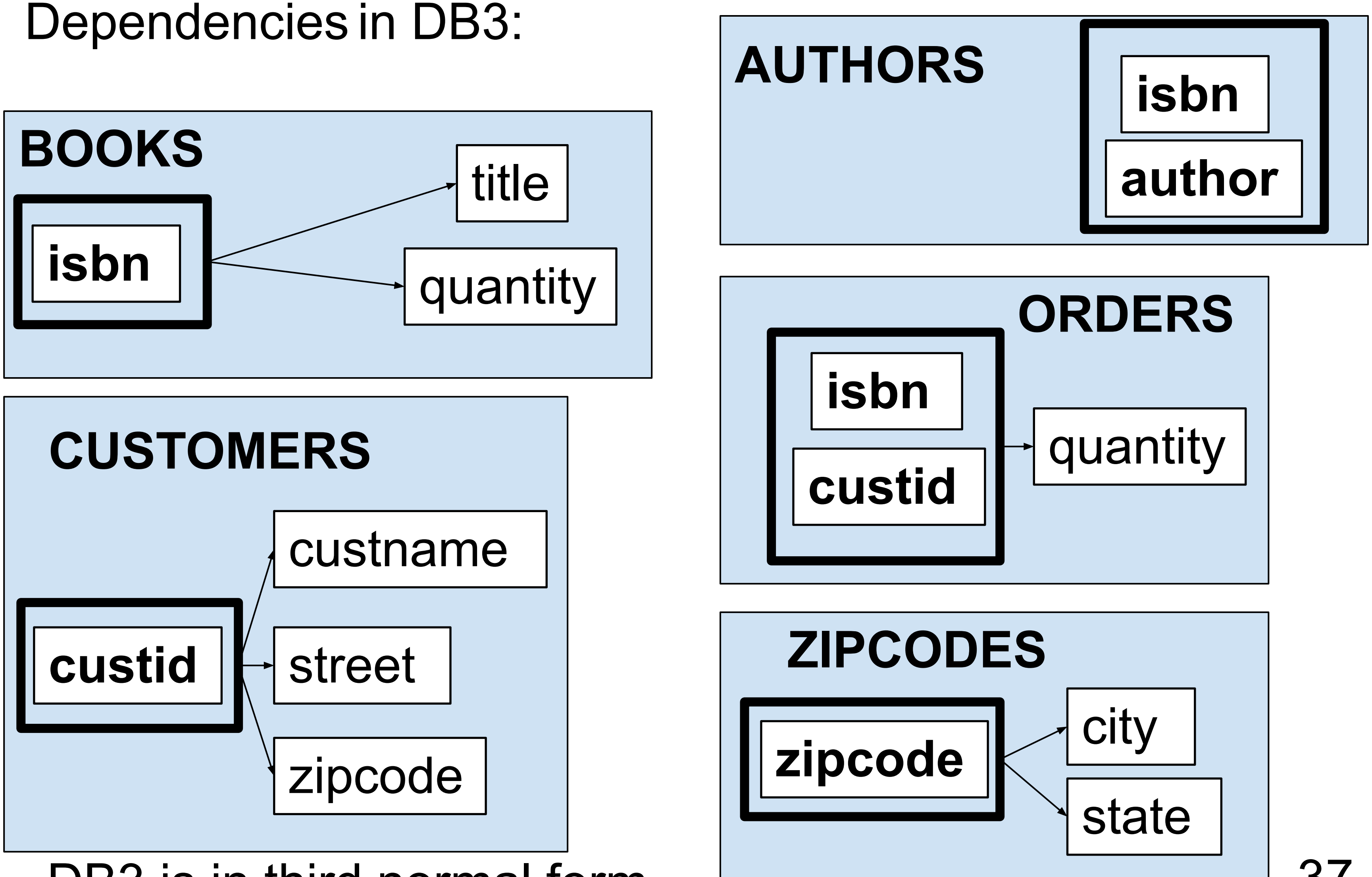
custid	custname	street	zipcode
111	Princeton	114 Nassau St	08540
222	Harvard	1256 Mass Ave	02138
333	MIT	292 Main St	02142

ZIPCODES

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

Relational DB Design: DB3

Dependencies in DB3:



DB3 is in third normal form

Relational DB Design: Summary

Codd's 1971 Paper

Unnormalized form

↓ *Eliminate domains which have relations as elements*

First normal form

↓ *Eliminate non-full dependence of non-prime attributes on candidate keys*

Second normal form

↓ *Eliminate transitive dependence of non-prime attributes on candidate keys*

Third normal form

Relational DB Design

- Some additional points:
  - Database designers routinely violate normal forms
  - There is a substantial mathematical theory of relational database design
  - DBMS can enforce additional ***consistency constraints***
    - See **bookstorefancy.sql**

Summary

- We have covered:
  - Relational DB transactions: atomicity
  - Relational DB transactions: locking
  - Relational DB design

Summary

- We have covered:
  - **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
- See also...

See Also

- **Appendices**
  - **Appendix 1**: Before relational DBs
  - **Appendix 2:** After relational DBs
- **Optional lecture slide decks**
  - PostgreSQL
  - SQLAlchemy

Appendix 1:
Before Relational DBs

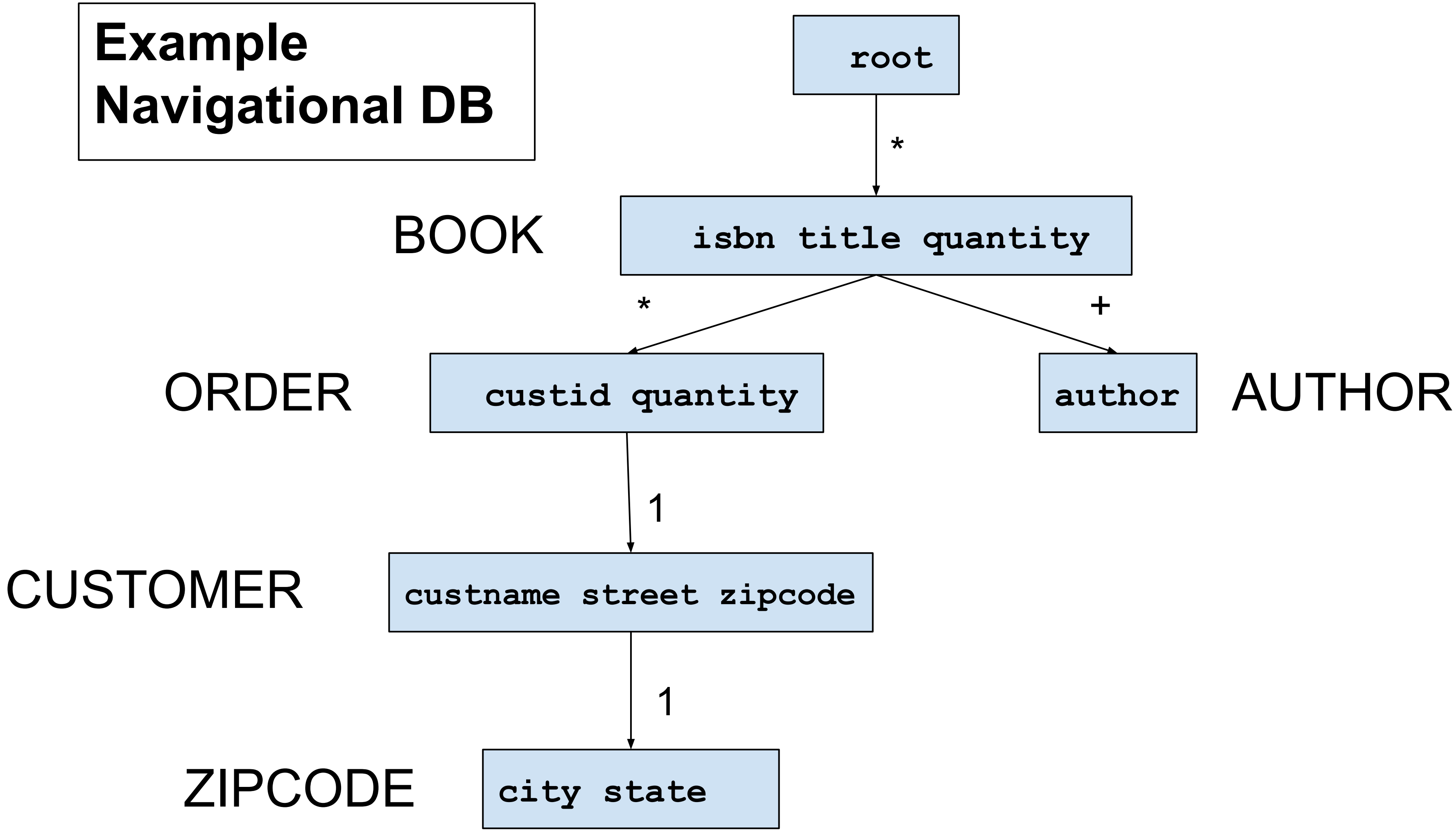
Before Relational DBs

- Before relational DBs, there were…

- ***Navigational*** DBs
  - Data are linked into graph structure

Before Relational DBs

Example Navigational DB



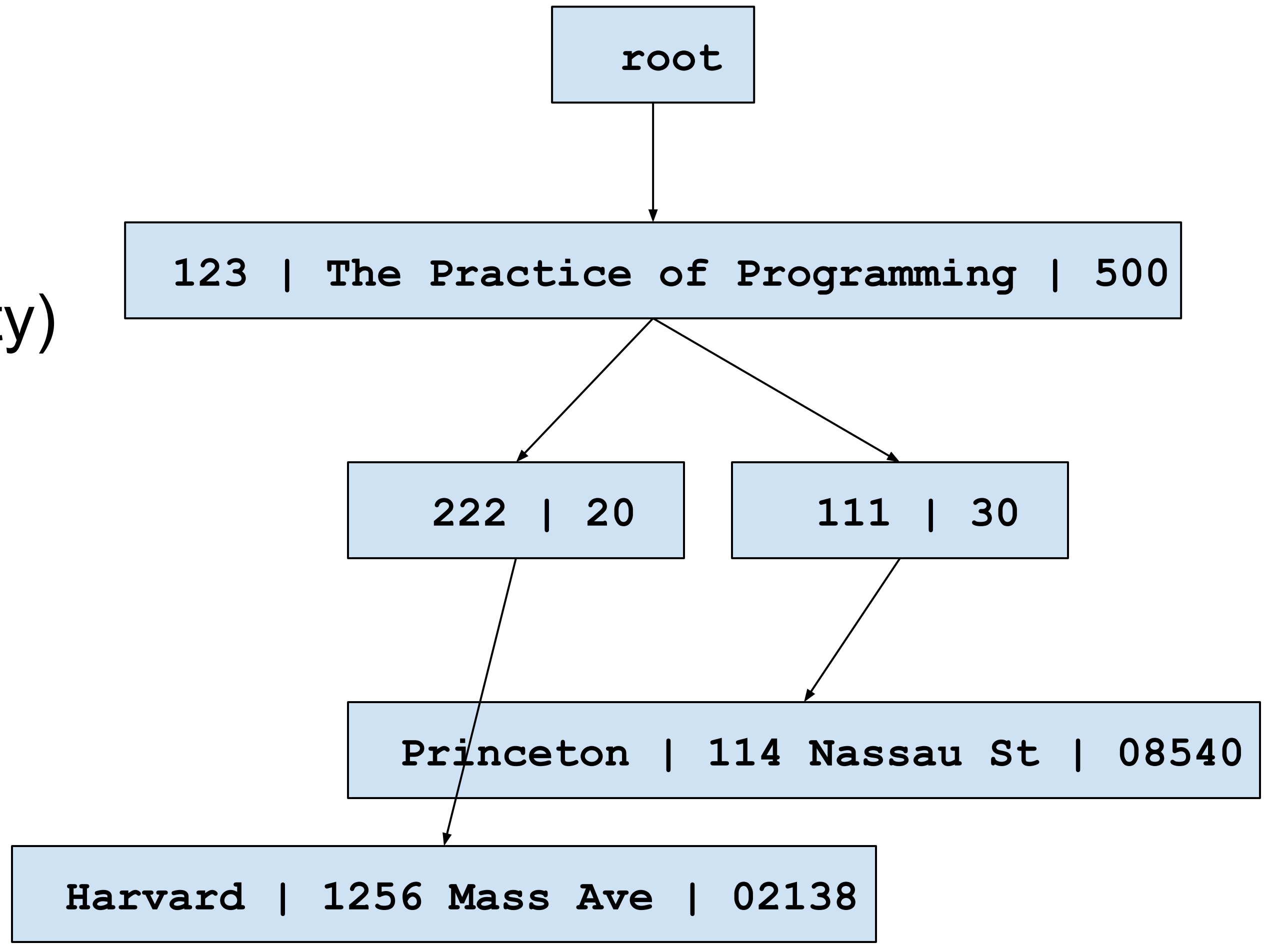
Before Relational DBs

Which customers purchased the book whose ISBN is 123?

BOOK
(isbn, title, quantity)

ORDER
(custid, quantity)

CUSTOMER
(custname, street, zipcode)



Before Relational DBs

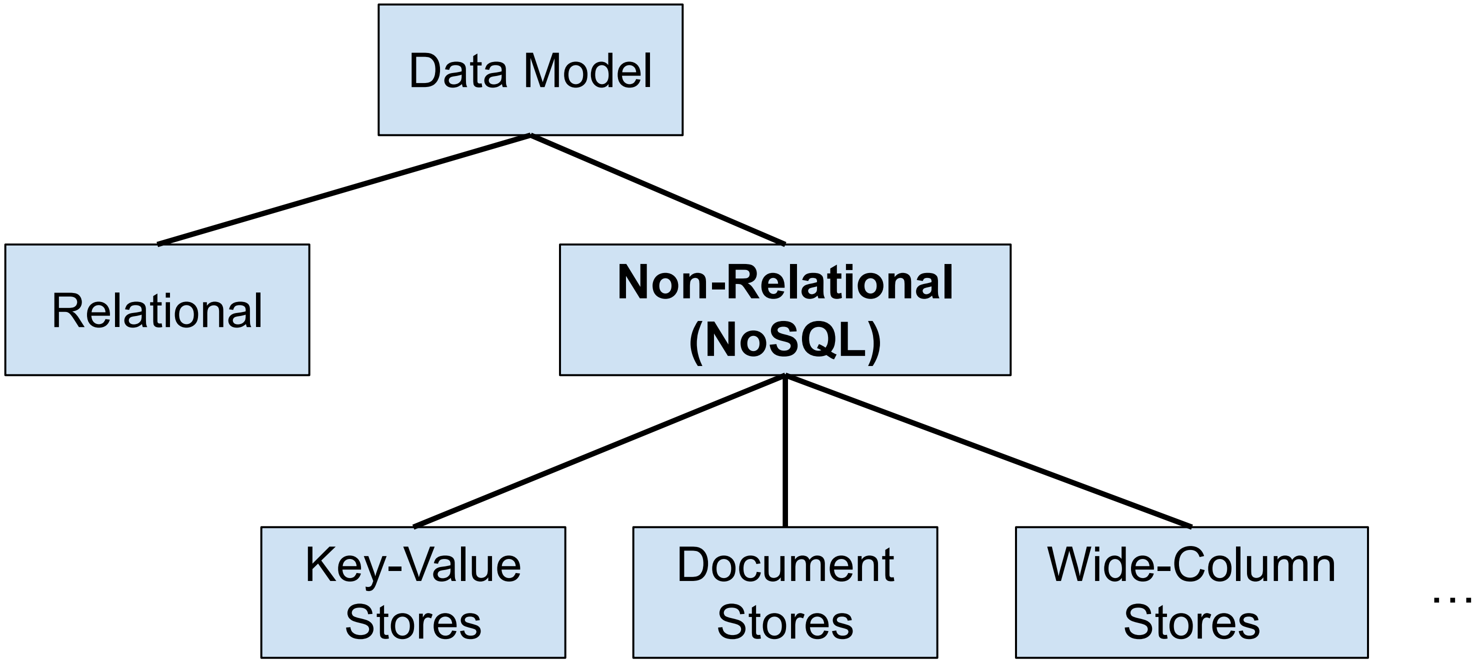
- Navigational DBs
  - Queries are **biased**
  - DB designer must anticipate queries
- Relational DBs
  - Queries are **unbiased**
  - DB designer need not anticipate queries
  - However, DB designer can create indices

Appendix 2:
After Relational DBs

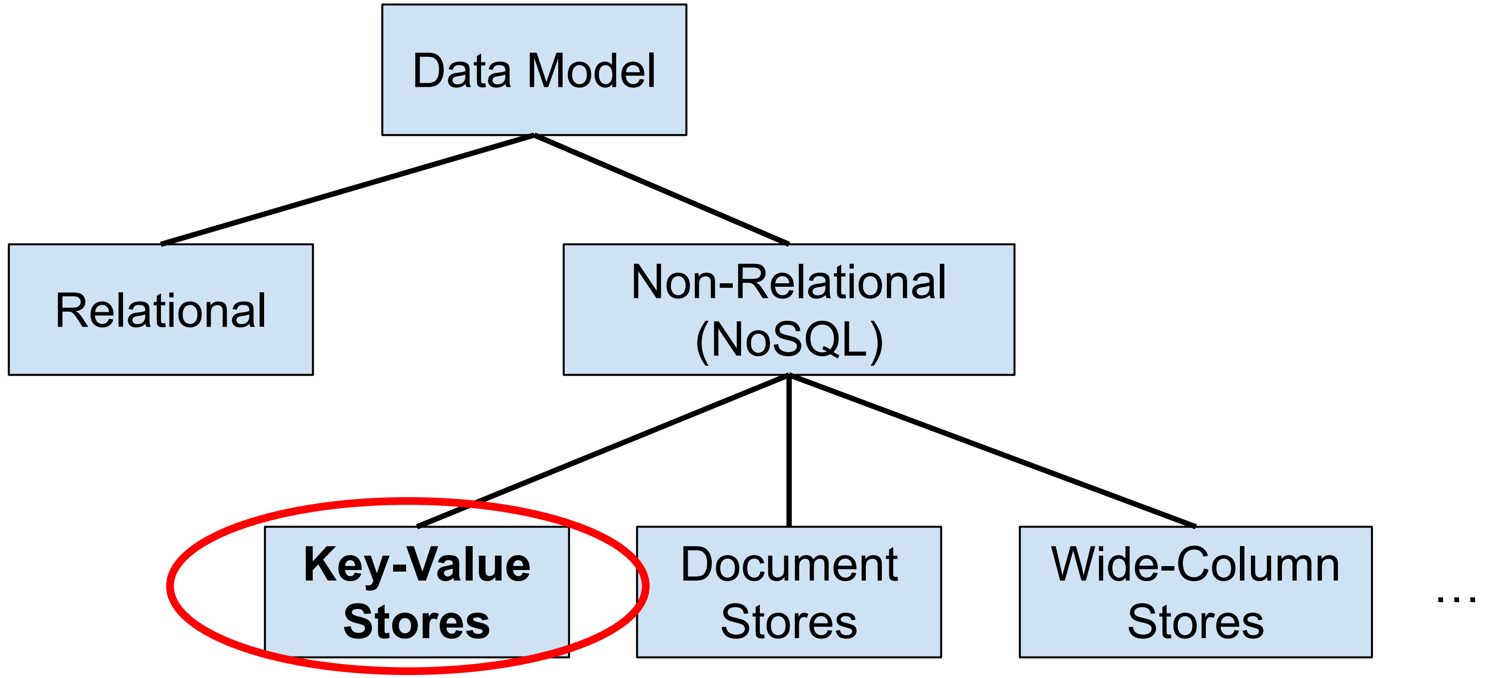
After Relational DBs

- For some apps:
  - Relational DBMSs are more complex than necessary
  - The relational DB model is a poor fit

After Relational DBs



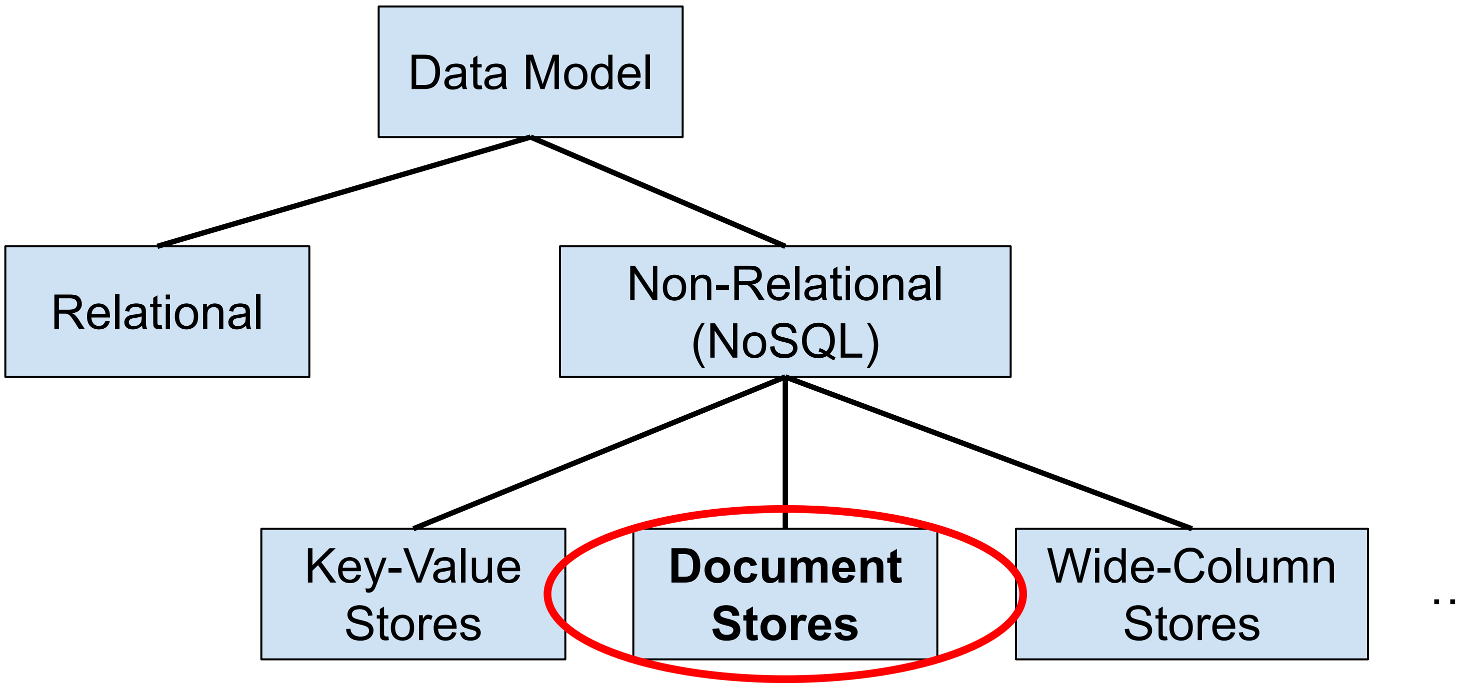
After Relational DBs



After Relational DBs

- ***Key-value store***
  - **Values**: arbitrary bytes
  - **Data structure**: key-value pairs
  - **Access**: by key
  - **Examples**: **Redis**, Memcached, Microsoft Azure Cosmos DB, Hazelcast, Ehcache

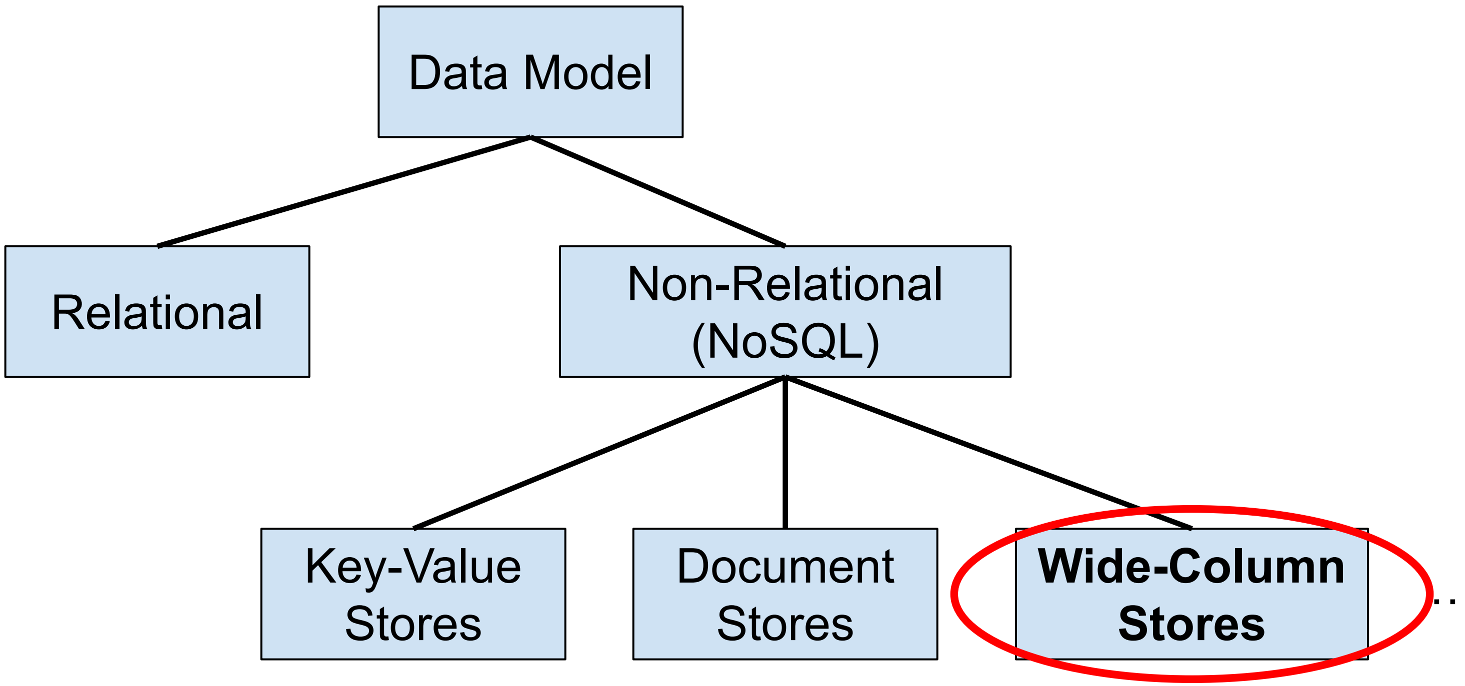
After Relational DBs



After Relational DBs

- ***Document store***
  - **Values**: documents with internal structure (e.g., JSON)
  - **Data structure**: key-value pairs
  - **Access**: by key or content
  - **Examples**: **MongoDB**, Amazon DynamoDB, Couchbase, CouchDB, MarkLogic

After Relational DBs



After Relational DBs

- ***Wide-column store***
  - **Values**: Arbitrary bytes
  - **Data structure**: Multidimensional associative array
  - **Examples**: **Cassandra**, HBase, Microsoft Azure Cosmos DB

After Relational DBs

Popular DBMSs, according to https://db-engines.com/en/ranking as of July 2024:

Rank	DBMS	DB Data Model	Score
1	Oracle	Relational	1258
2	MySQL	Relational	1027
3	Microsoft SQL Server	Relational	815
4	PostgreSQL	Relational	637
5	MongoDB	Document Store	421
6	Redis	Key-Value Store	153
…	…	…	…
10	SQLite	Relational	105