

# [544] SQL Databases (MySQL)

Meenakshi Syamkumar

# Learning Objectives

- create database schemas with types and keys
- use database transactions to group multiple updates together
- write SQL queries with common clauses (SELECT, FROM, JOIN, WHERE, GROUP BY, HAVING, ORDER BY, and LIMIT) to answer questions about data

# Outline

## Creating/designing tables

- data modeling
- primary/foreign keys

Transactions

Queries

Demos

# Data Modeling

**Data modeling:** deciding how to represent something in an underlying system.

**Low-level example (protobufs):** how will we represent numbers as bytes being sent over a network?

**Traditional Databases:** how will we represent things/people/events/etc as rows in tables?

option 1:

tbl\_orders

name	book	amount	county	state
Tyler Harter	Designing Data-Intensive Applications	23	Dane	WI
Tyler Harter	Learning Spark	38	Dane	WI
Tyler Harter	Cassandra: The Definitive Guide	39	Dane	WI

# Keys and Normalization

SQL keys:

- **primary key**: uniquely identify a row ("id" in tbl\_counties)
- **foreign key**: reference a primary key ("county\_id" in tbl\_orders)

In database theory we would say option 2 is "more **normalized**" (note: there are well-defined normalization levels with formal rules -- we won't get into that in 544)

option 1:

tbl\_orders

name	book	amount	county	state
Tyler Harter	Designing Data-Intensive Applications	23	Dane	WI
Tyler Harter	Learning Spark	38	Dane	WI
Tyler Harter	Cassandra: The Definitive Guide	39	Dane	WI

option 2:

tbl_orders				tbl_counties		
name	book	amount	county_id	id	county	state
Tyler Harter	Designing Data-Intensive Applications	23	1	1	Dane	WI
Tyler Harter	Learning Spark	38	1	2	Milwaukee	WI
Tyler Harter	Cassandra: The Definitive Guide	39	1	3	La Crosse	WI

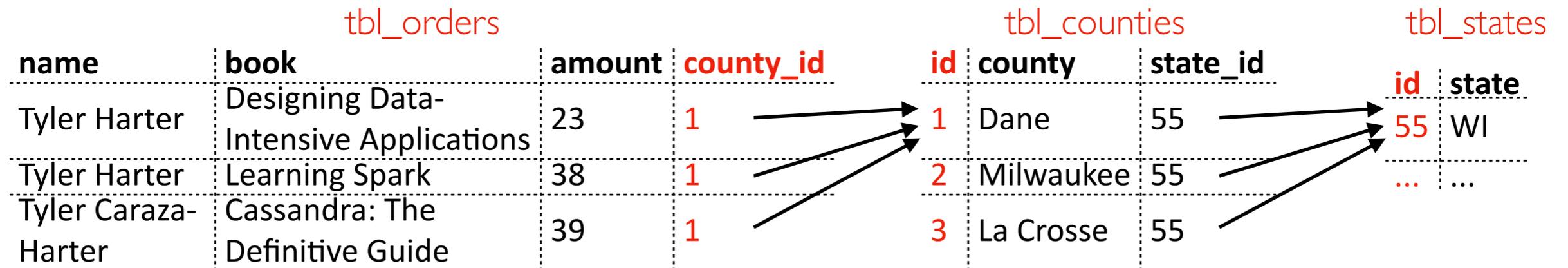
# Normalization Tradeoffs

## Benefits of more normalization:

- avoid inconsistencies
- changes in the real world correspond to fewer changes in the DB
- often save space

## Downsides of more normalization:

- queries are sometimes slower
- historical record keeping (for example, if you need to reproduce an invoice prior to somebody's name change, you might want the name at time of purchase)



# Outline

Creating/designing tables

- data modeling
- primary/foreign keys

Transactions

Queries

Demos

# Definitions of Transactions

Definition 1, regarding access patterns

- **analytics**: calculate over many/all rows, few columns (corresponding DB: OLAP)
- **transactions**: work with whole row or few rows at a time (corresponding DB: OLTP)

Definition 2, regarding guarantees for a collection of DB operations (often changes).

Common guarantees:

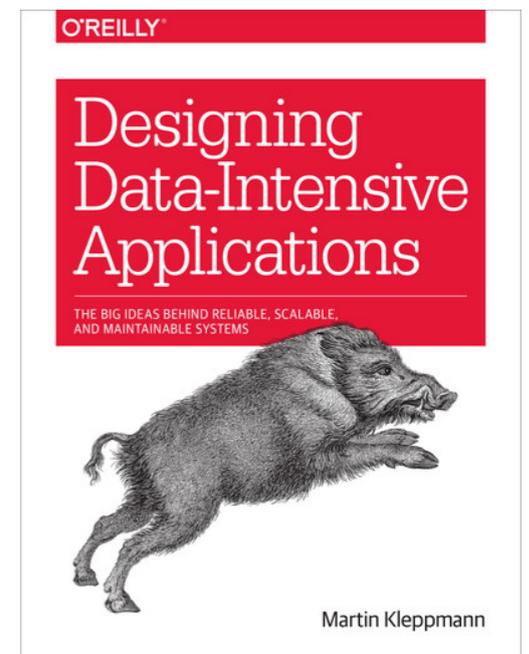
- **atomicity**: it all happens or nothing happens (partial progress is rolled back upon failure)
- **consistency**: application invariants (like no negative bank accounts) are supported
- **isolation**: others cannot see a transaction in progress (aka **atomicity** when talking about locks)
- **durability**: once finished, it persists (even if machine crashes+restarts)

Transactions in a DB are similar to critical sections in a multi-threaded process:

```
8  if bank_accounts[src] >= dollars:  
9      bank_accounts[src] -= dollars  
10     bank_accounts[dst] += dollars
```

critical section  
(example from "locks" lecture)

"NoSQL" databases often have weaker transactions (not ACID) in order to achieve other goals (e.g., performance, scalability, availability, etc).



"The Meaning of ACID"

# Outline

Creating/designing tables

- data modeling
- primary/foreign keys

Transactions

Queries

Demos

# SQL Query: General Structure

SELECT

FROM

*JOIN (optional)*

*WHERE (optional)*

*GROUP BY (optional)*

*HAVING (optional)*

*ORDER BY (optional)*

*LIMIT (optional)*

;



# Outline

Creating/designing tables

- data modeling
- primary/foreign keys

Transactions

Queries

Demos

# Banking Demos

