Chapter 8

对冗余的控制（反范式模式）

Consider Introduction of Controlled Redundancy

Consider the Introduction of Controlled Redundancy

- Determine whether introducing redundancy in a controlled manner by relaxing the normalization rules will improve system performance.

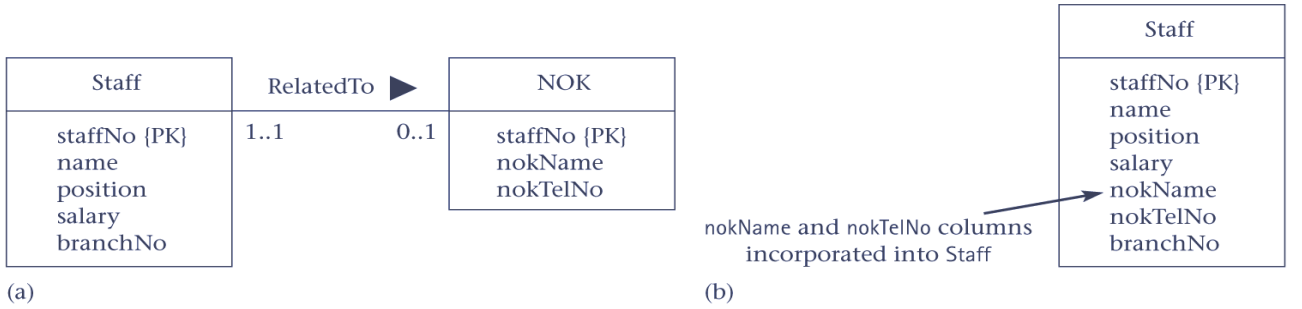
Denormalization

- Refinement to relational schema such that the degree of normalization for a modified table is less than the degree of at least one of the original tables.
- Also use term more loosely to refer to situations where two tables are combined into one new table, which is still normalized but contains more nulls than original tables.

Consider the Introduction of Controlled Redundancy

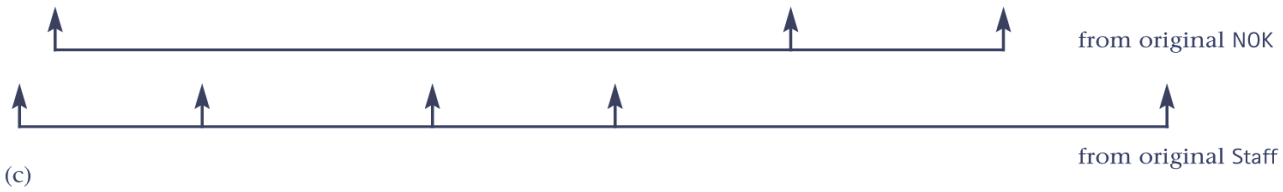
- Consider denormalization in following situations, specifically to speed up frequent or critical transactions:
  - Pattern 1 Combining 1:1 relationships
  - Pattern 2 Duplicating nonkey columns in 1:* relationships to reduce joins
  - Pattern 3 Duplicating FK columns in 1:* relationships to reduce joins
  - Pattern 4 Duplicating columns in *:* relationships to reduce joins
  - Pattern 5 Introducing repeating groups
  - Pattern 6 Creating extract tables
  - Pattern 7 Partitioning tables.

Pattern 1 Combining 1:1 relationships

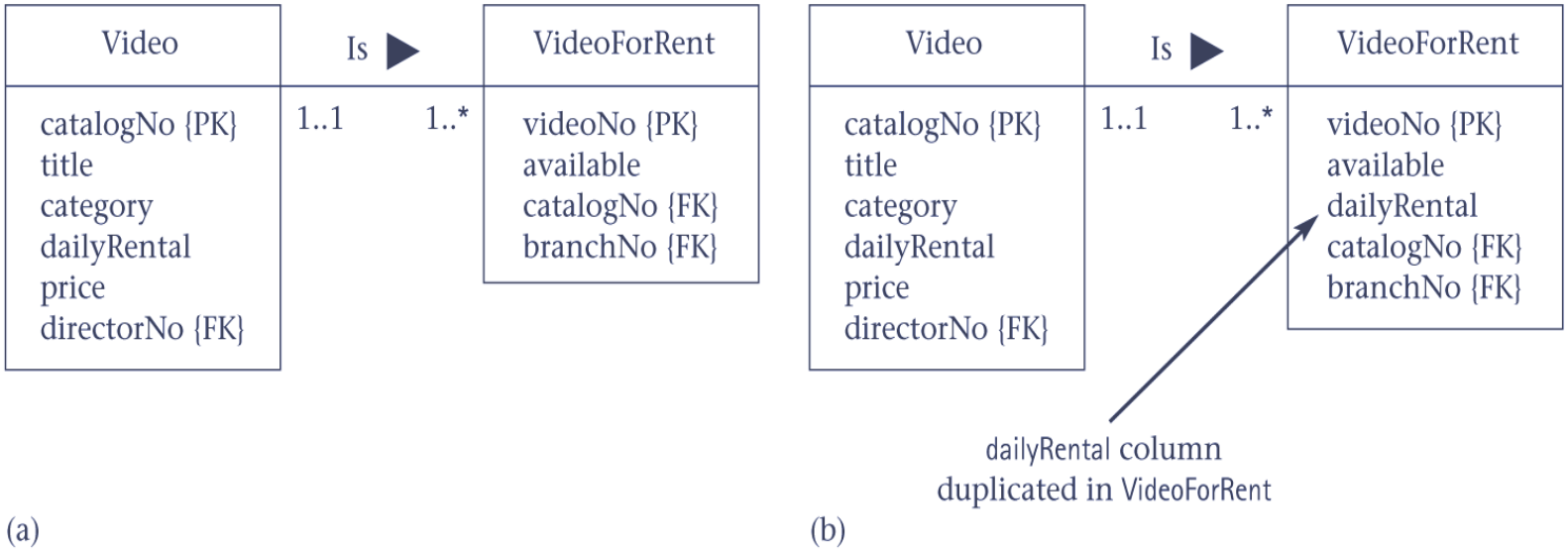


Staff

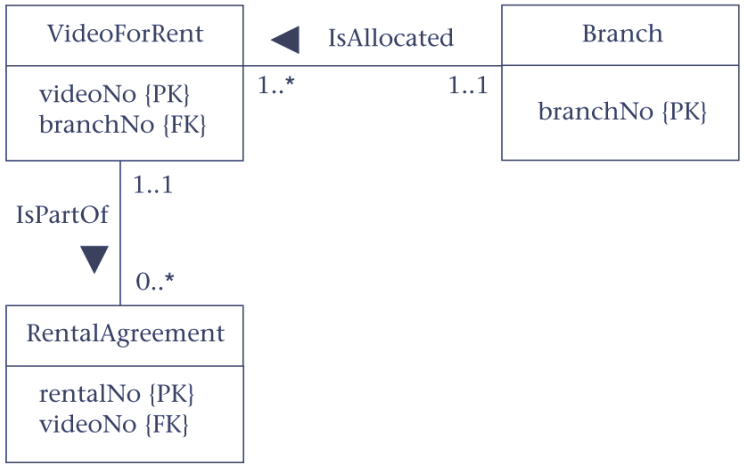
staffNo	name	position	salary	nokName	nokTelNo	branchNo
S1500	Tom Daniels	Manager	46000	Jane Daniels	207-878-2751	B001
S0003	Sally Adams	Assistant	30000	John Adams	518-474-5355	B001
S0010	Mary Martinez	Manager	50000			B002
S3250	Robert Chin	Supervisor	32000	Michelle Chin	206-655-9867	B002
S2250	Sally Stern	Manager	48000			B004
S0415	Art Peters	Manager	41000	Amy Peters	718-507-7923	B003



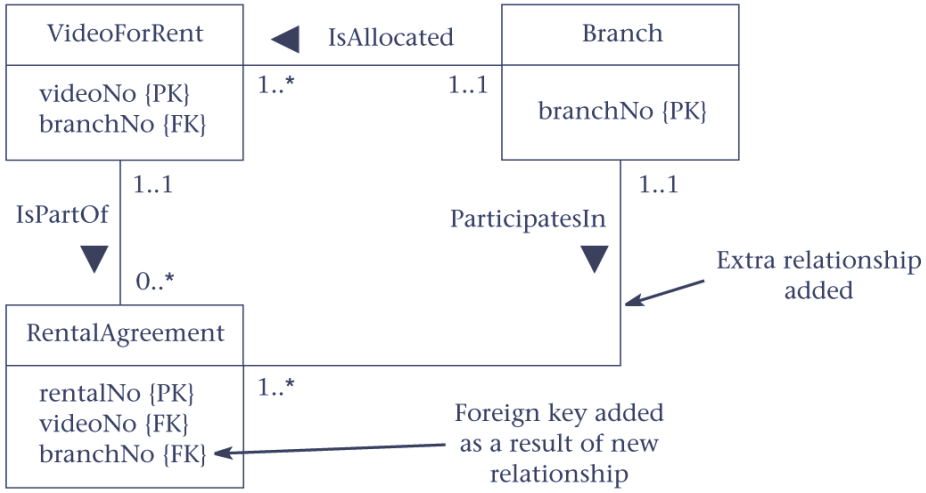
Pattern 2 Duplicating nonkey columns in 1:* relationships to reduce joins



Pattern 3 Duplicating FK columns in 1:* relationship to reduce joins

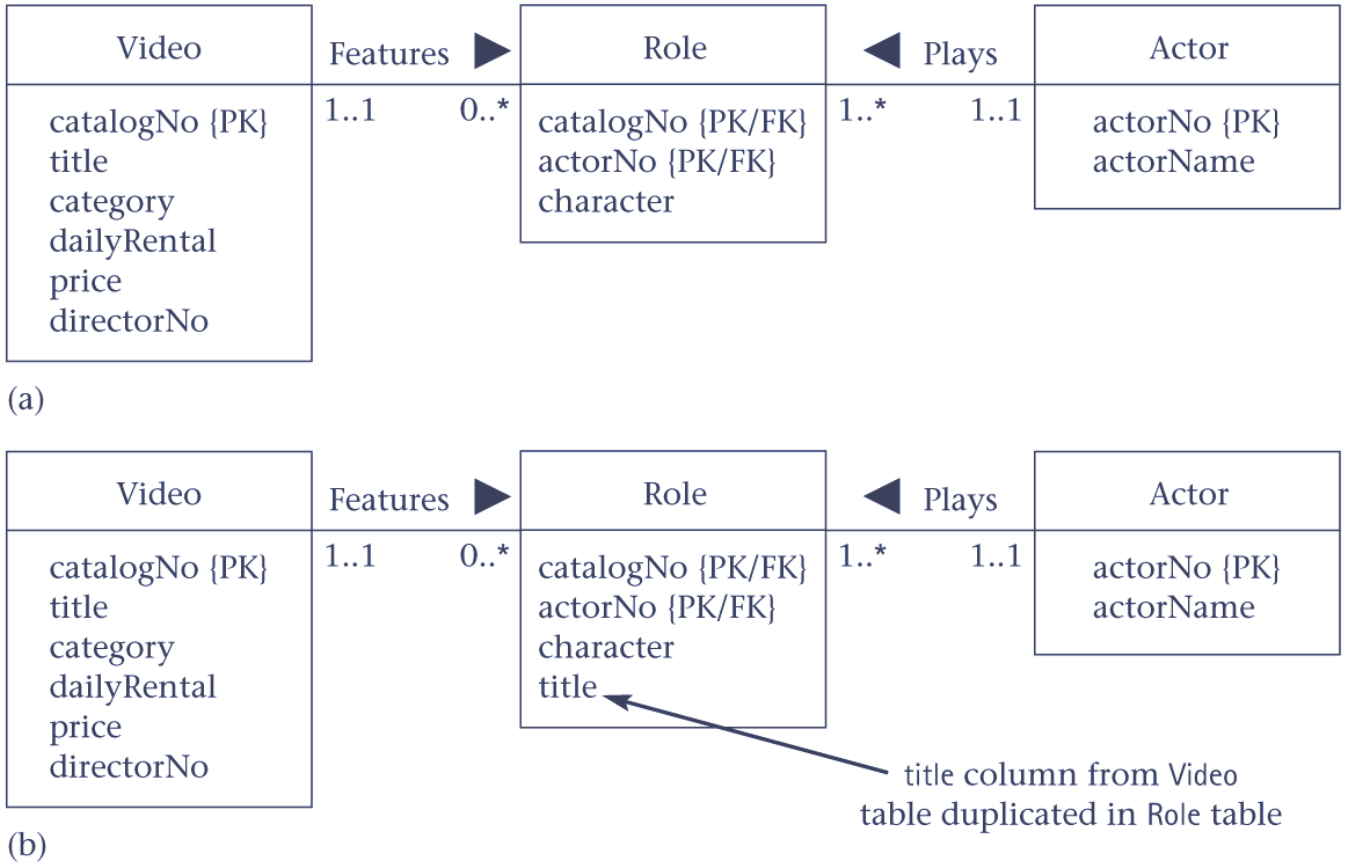


(a)



(b)

Pattern 4 Duplicating columns in *:* relationships to reduce joins



Pattern 5 Introducing repeating groups

Pattern 6 Creating extract tables

- Reports can access derived data and perform multi-table joins on same set of base tables. However, data report based on may be relatively static or may not have to be current.
- Can create a single, highly denormalized extract table based on tables required by reports, and allow users to access extract table directly instead of base tables.

Pattern 7 Partitioning tables

- Rather than combining tables, could decompose a table into a smaller number of partitions.
- Horizontal partition: distribute records across a number of (smaller) tables.
- Vertical partition: distribute columns across a number of (smaller) tables. PK duplicated to allow reconstruction.
- Partitions useful for applications that store and analyze large amounts of data.

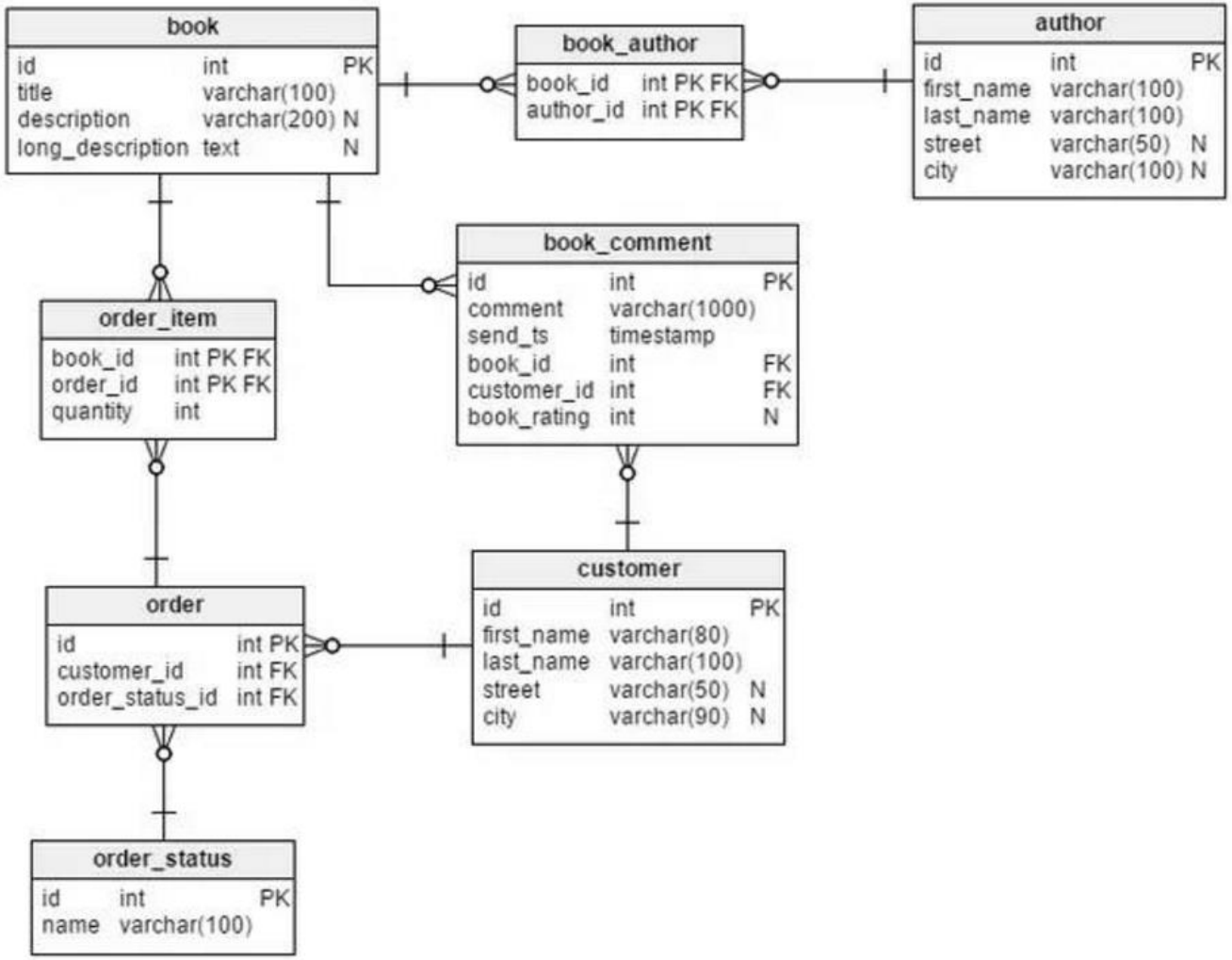
Chapter 9

一个示例，看数据库设计通常会出的问题

模型设计

- 假设我们想要为一个在线书城设计数据库。该系统应当允许用户执行以下活动：
  - 通过书名、描述和作者信息浏览与搜索书籍，
  - 阅读后对书籍添加评论和评级，
  - 定购书籍，
  - 查看订单处理的状态。

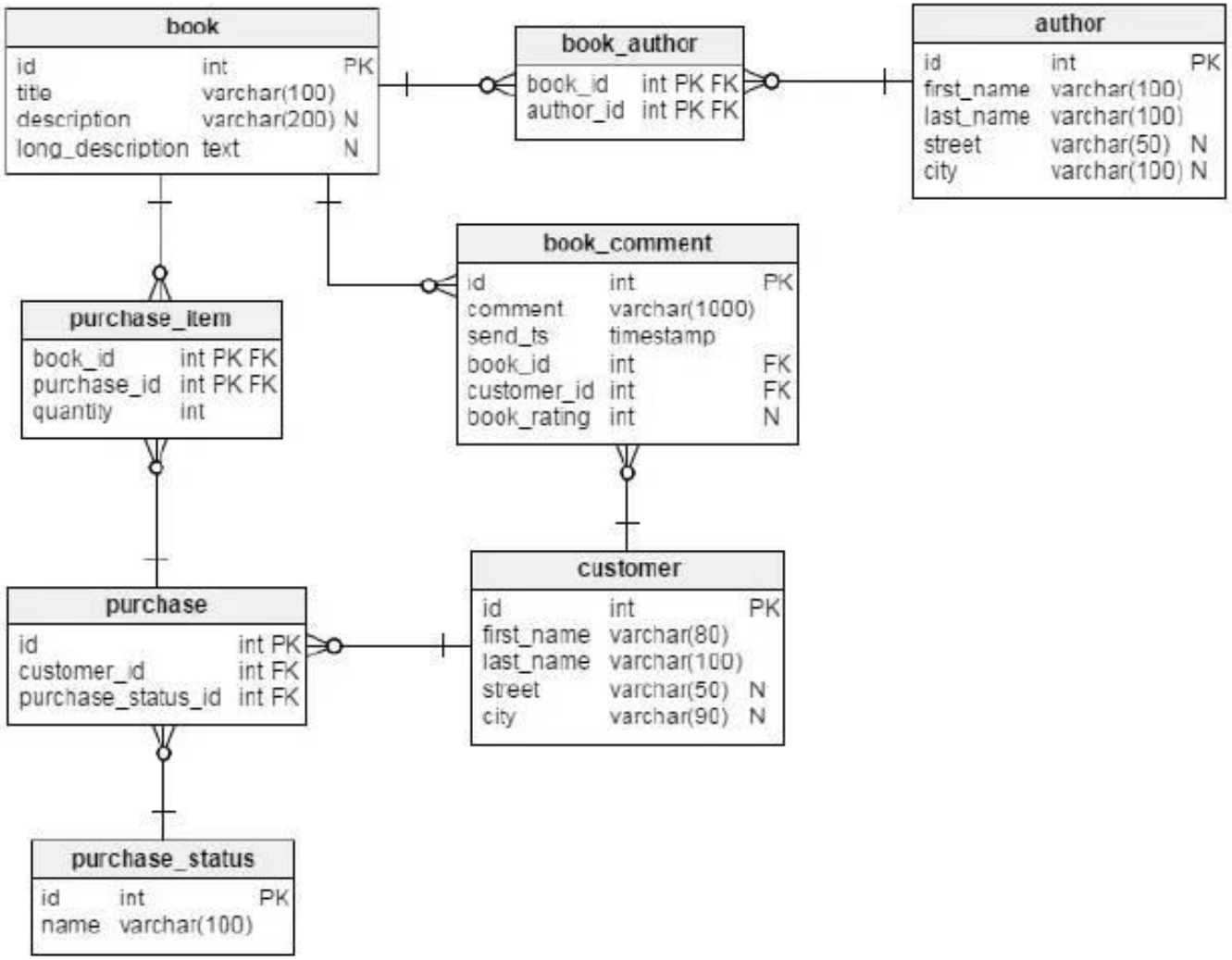
模型-0.1


book
id int PK
title varchar(100)
description varchar(200) N
long_description text N
book_author
book_id int PK FK
author_id int PK FK
author
id int PK
first_name varchar(100)
last_name varchar(100)
street varchar(50) N
city varchar(100) N
book_comment
id int PK
comment varchar(1000)
send_ts timestamp
book_id int FK
customer_id int FK
book_rating int N
order_item
book_id int PK FK
order_id int PK FK
quantity int
order
id int PK
customer_id int FK
order_status_id int FK
customer
id int PK
first_name varchar(80)
last_name varchar(100)
street varchar(50) N
city varchar(90) N
order_status
id int PK
name varchar(100)

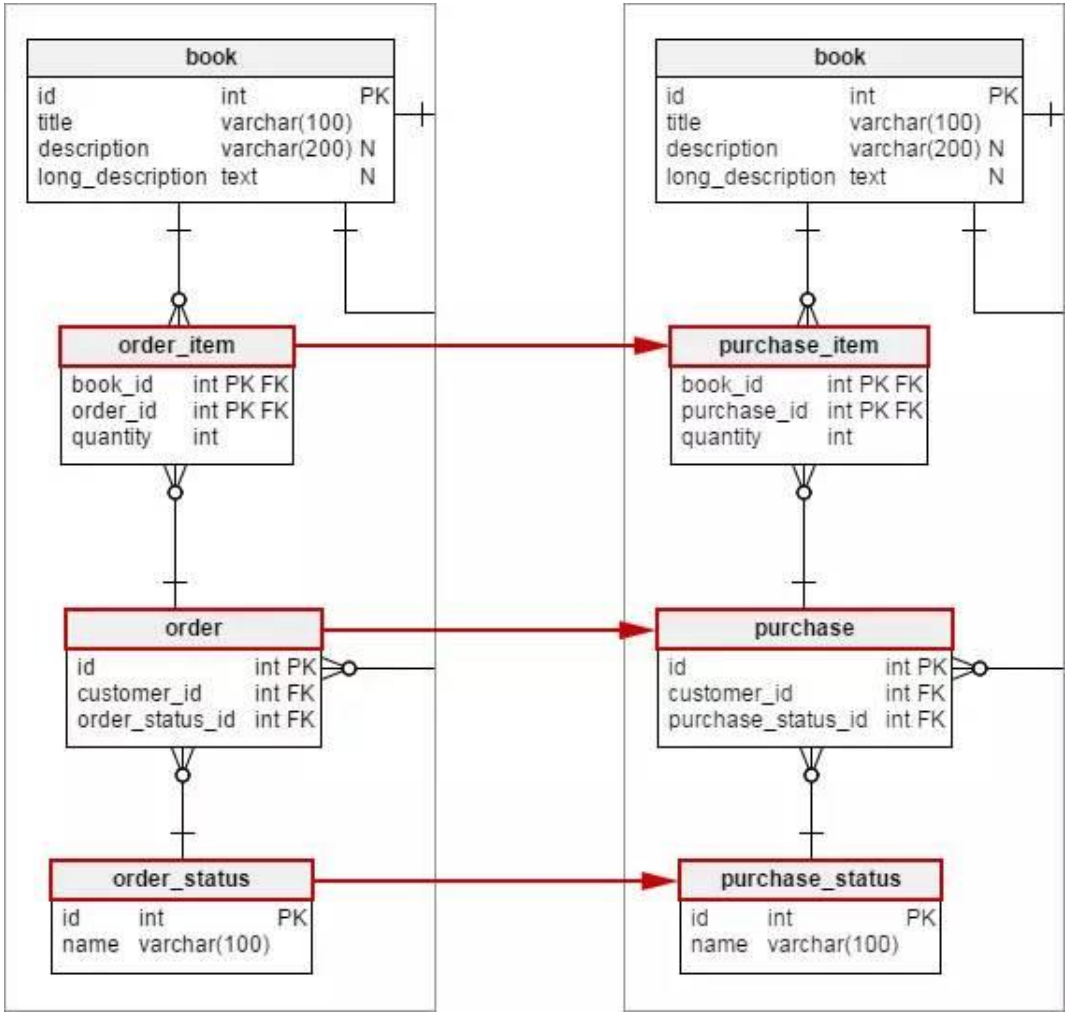
问题1：使用无效的名称

- 底线：不要使用关键字来当做对象名称，NEVER
- 提示
  - 让你的数据库中的名字：
    - 尽可能短，
    - 直观，尽可能正确和具有描述性，
    - 保持一致性；
  - 避免使用SQL和数据库引擎特定的关键字作为名字；
  - 建立命名约定；

模型-0.2


book
id int PK
title varchar(100)
description varchar(200) N
long_description text N
book_author
book_id int PK FK
author_id int PK FK
author
id int PK
first_name varchar(100)
last_name varchar(100)
street varchar(50) N
city varchar(100) N
book_comment
id int PK
comment varchar(1000)
send_ts timestamp
book_id int FK
customer_id int FK
book_rating int N
purchase_item
book_id int PK FK
purchase_id int PK FK
quantity int
purchase
id int PK
customer_id int FK
purchase_status_id int FK
customer
id int PK
first_name varchar(80)
last_name varchar(100)
street varchar(50) N
city varchar(90) N
purchase_status
id int PK
name varchar(100)

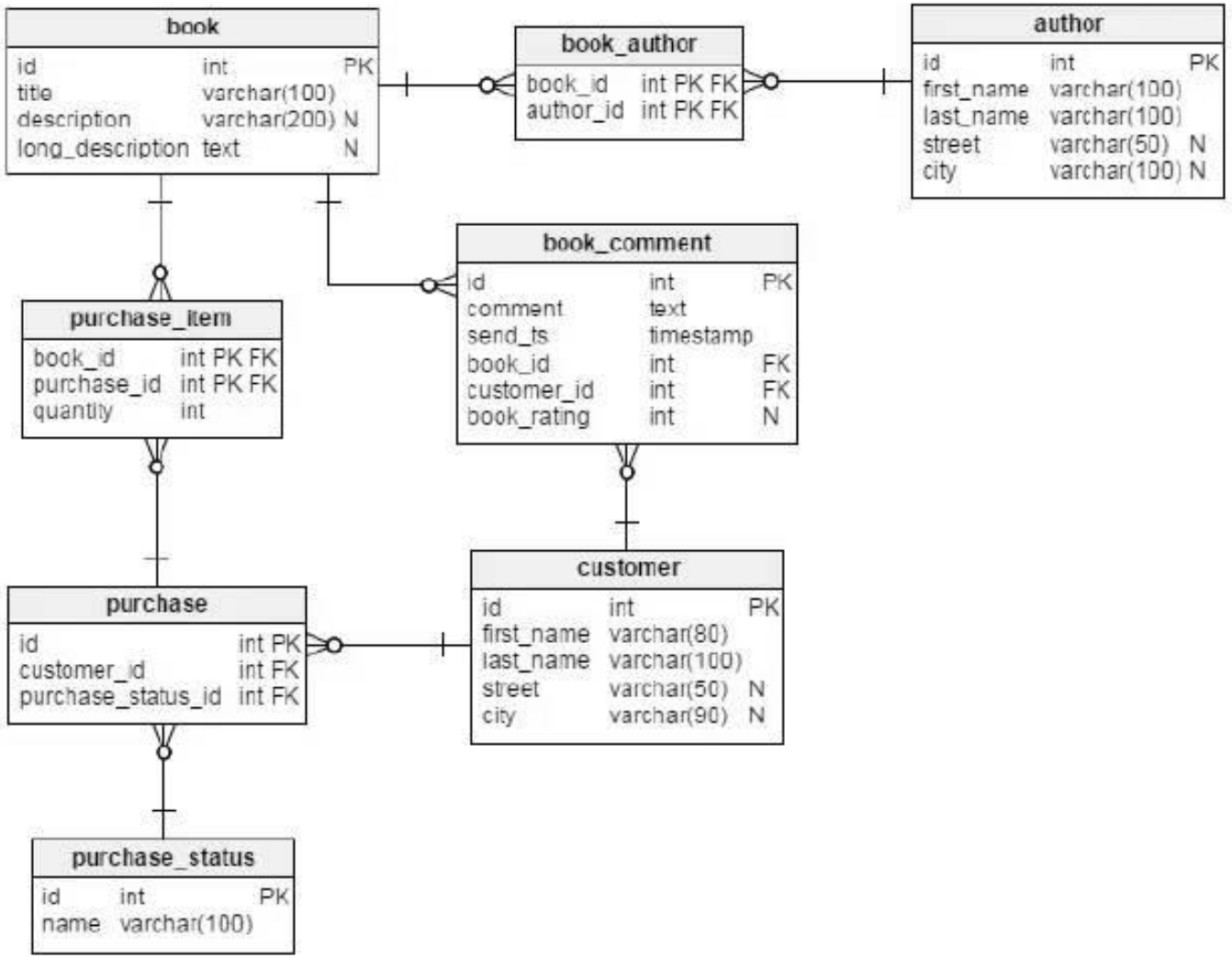
模型-0.2（detail）


book
id int PK
title varchar(100)
description varchar(200) N
long_description text N
order_item
book_id int PK FK
order_id int PK FK
quantity int
order
id int PK
customer_id int FK
order_status_id int FK
order_status
id int PK
name varchar(100)
book
id int PK
title varchar(100)
description varchar(200) N
long_description text N
purchase_item
book_id int PK FK
purchase_id int PK FK
quantity int
purchase
id int PK
customer_id int FK
purchase_status_id int FK
purchase_status
id int PK
name varchar(100)

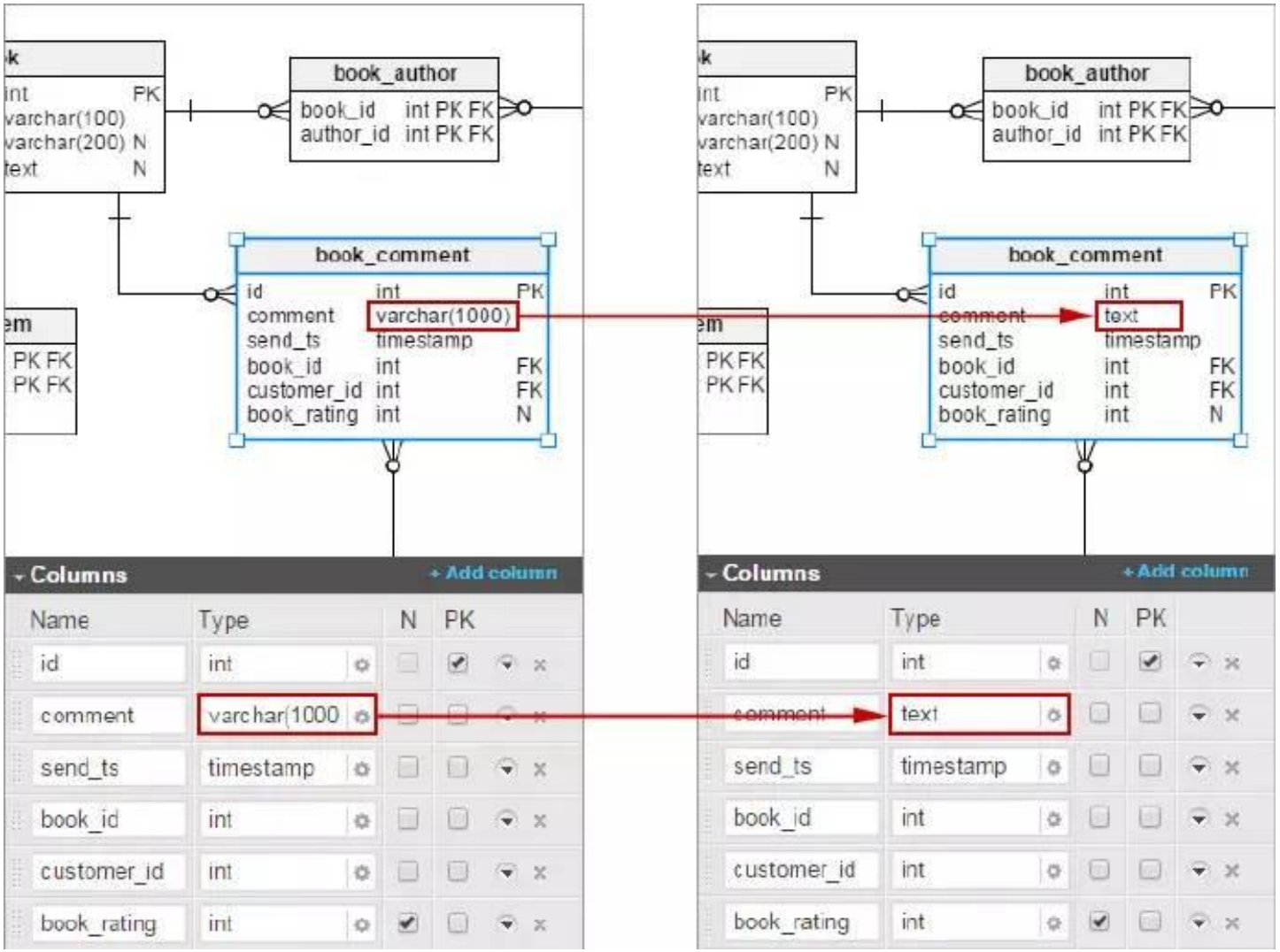
问题2：列的宽度不足

- 文本字段的限制以及应当始终谨记文本的编码方式
  - varchar(100)-PostgreSQL：100个字符，Oracle：100字节。
- 不同数据库对于可变长的字符和文本字段会有不同的限制
- 提示：
  - 一般而言，考虑到安全和性能，数据库中限制文本列的长度是好的，但有时这个做法可能没有必要或者不方便；
  - 不同的数据库对待文本限制可能会有差异；
  - 使用英语以外的语言时永远记住编码。

模型-0.3


book
id int PK
title varchar(100)
description varchar(200) N
long_description text N
book_author
book_id int PK FK
author_id int PK FK
author
id int PK
first_name varchar(100)
last_name varchar(100)
street varchar(50) N
city varchar(100) N
book_comment
id int PK
comment text
send_ts timestamp
book_id int FK
customer_id int FK
book_rating int N
purchase_item
book_id int PK FK
purchase_id int PK FK
quantity int
purchase
id int PK
customer_id int FK
purchase_status_id int FK
customer
id int PK
first_name varchar(80)
last_name varchar(100)
street varchar(50) N
city varchar(90) N
purchase_status
id int PK
name varchar(100)

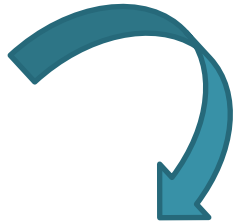
模型-0.3（detail）


book_author
book_id int PK FK
author_id int PK FK
book_comment
id int PK
comment varchar(1000)
send_ts timestamp
book_id int FK
customer_id int FK
book_rating int N
Columns
+ Add column
Name Type N PK
id int
comment varchar(1000
send_ts timestamp
book_id int
customer_id int
book_rating int
book_author
book_id int PK FK
author_id int PK FK
book_comment
id int PK
comment text
send_ts timestamp
book_id int FK
customer_id int FK
book_rating int N
Columns
+ Add column
Name Type N PK
id int
comment text
send_ts timestamp
book_id int
customer_id int
book_rating int

问题3：没有恰当地添加索引

- 在首页显示最新的30条评论
- select comment, send_ts from book_comment order by send_ts desc limit 30;

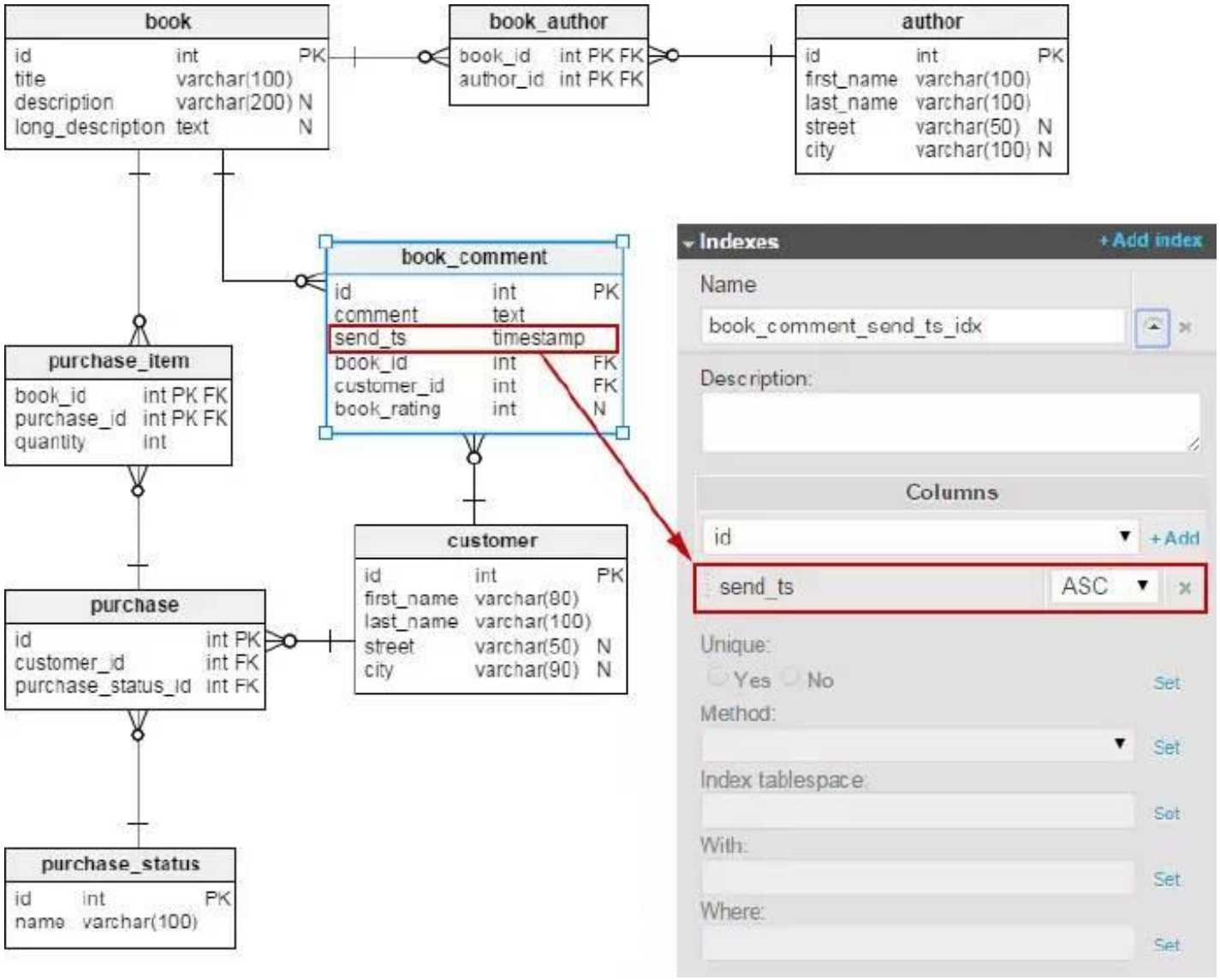
```
db=# explain select comment, send_ts from book_comment
order by send_ts desc limit 30;
QUERY PLAN
------------------------------------------------------------------
Limit (cost=28244.01..28244.09 rows=30 width=17)
-> Sort (cost=28244.01..29751.62 rows=603044 width=17)
Sort Key: send_ts
-> Seq Scan on book_comment (cost=0.00..10433.44
rows=603044 width=17)
```



create index book_comment_send_ts_idx on book_comment(send_ts);

```
db=# explain select comment, send_ts from book_comment
order by send_ts desc limit 30;
QUERY PLAN
------------------------------------------------------------
------------
Limit (cost=0.42..1.43 rows=30 width=17)
-> Index Scan Backward using book_comment_send_ts_idx on
book_comment (cost=0.42..20465.77 rows=610667 width=17)
```

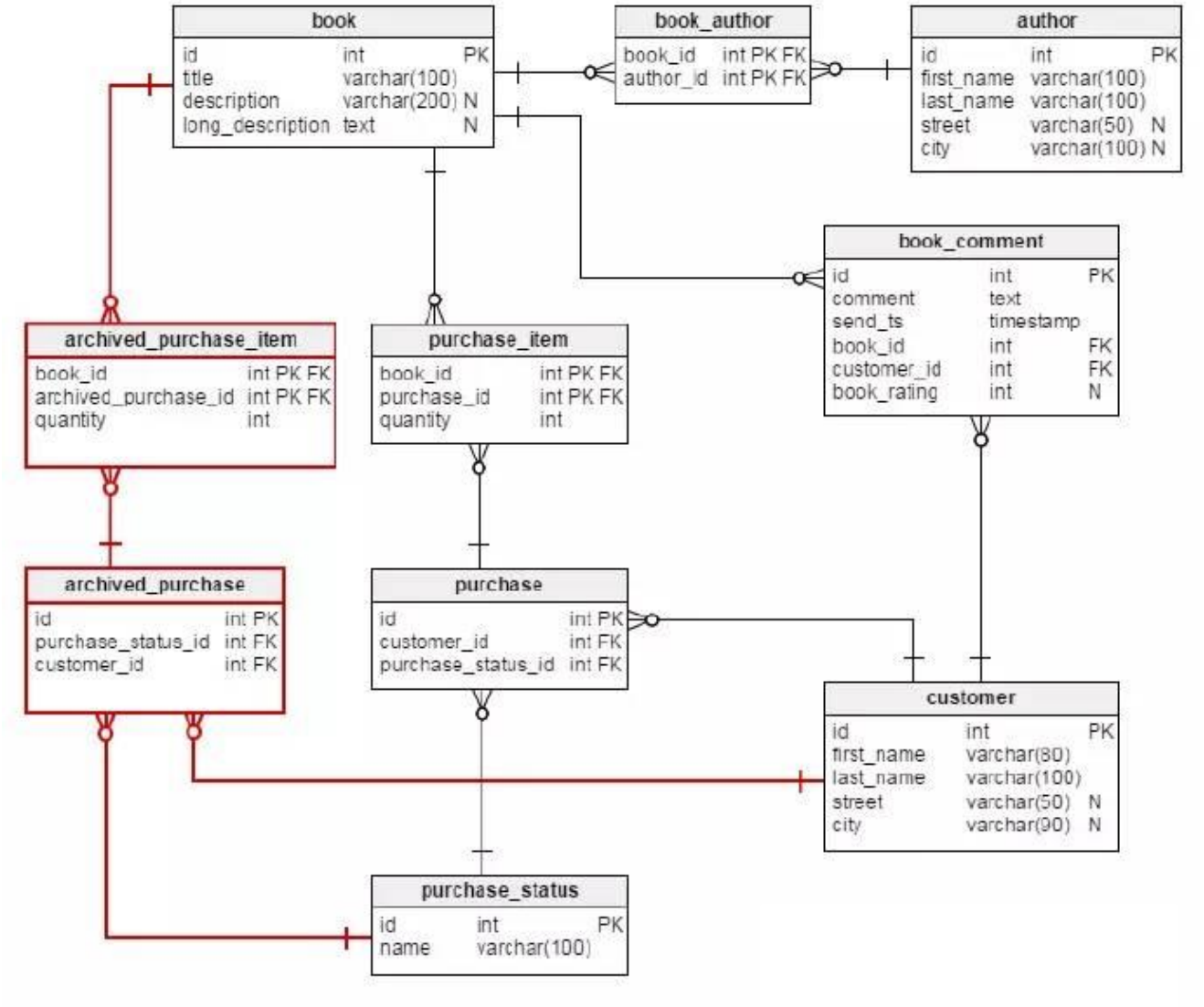
模型-0.4


book
id int PK
title varchar(100)
description varchar(200) N
long_description text N
book_author
book_id int PK FK
author_id int PK FK
author
id int PK
first_name varchar(100)
last_name varchar(100)
street varchar(50) N
city varchar(100) N
book_comment
id int PK
comment text
send_ts timestamp
book_id int FK
customer_id int FK
book_rating int N
purchase_item
book_id int PK FK
purchase_id int PK FK
quantity int
customer
id int PK
first_name varchar(80)
last_name varchar(100)
street varchar(50) N
city varchar(90) N
purchase
id int PK
customer_id int FK
purchase_status_id int FK
purchase_status
id int PK
name varchar(100)
Indexes
+ Add index
Name
book_comment_send_ts_idx
Description:
Columns
id
+ Add
send_ts
ASC
Unique:
Yes No
Set
Method:
Set
Index tablespace:
Set
With:
Set
Where:
Set

问题4：没有考虑到可能的数据量或流量

- 分离频繁和不频繁使用的数据到多个表中
  - purchase表的数据量可能会非常大
    - 分表：purchase表记录当前的订单，archived_purchase表记录完成的订单
  - 类似的，应当优化频繁更新的数据（基本信息和奖励积分）
  - 应用级别的缓存
- 提示：
  - 你的客户必须使用业务、领域特定的知识，预估预期你将处理的数据库中的数据量。
  - 分离频繁更新和频繁读取的数据。
  - 对重量级、更新少的数据考虑使用应用级别的缓存。

模型-0.5


book
id int PK
title varchar(100)
description varchar(200) N
long_description text N
book_author
book_id int PK FK
author_id int PK FK
author
id int PK
first_name varchar(100)
last_name varchar(100)
street varchar(50) N
city varchar(100) N
book_comment
id int PK
comment text
send_ts timestamp
book_id int FK
customer_id int FK
book_rating int N
archived_purchase_item
book_id int PK FK
archived_purchase_id int PK FK
quantity int
purchase_item
book_id int PK FK
purchase_id int PK FK
quantity int
archived_purchase
id int PK
purchase_status_id int FK
customer_id int FK
purchase
id int PK
customer_id int FK
purchase_status_id int FK
customer
id int PK
first_name varchar(80)
last_name varchar(100)
street varchar(50) N
city varchar(90) N
purchase_status
id int PK
name varchar(100)

问题5：忽略时区

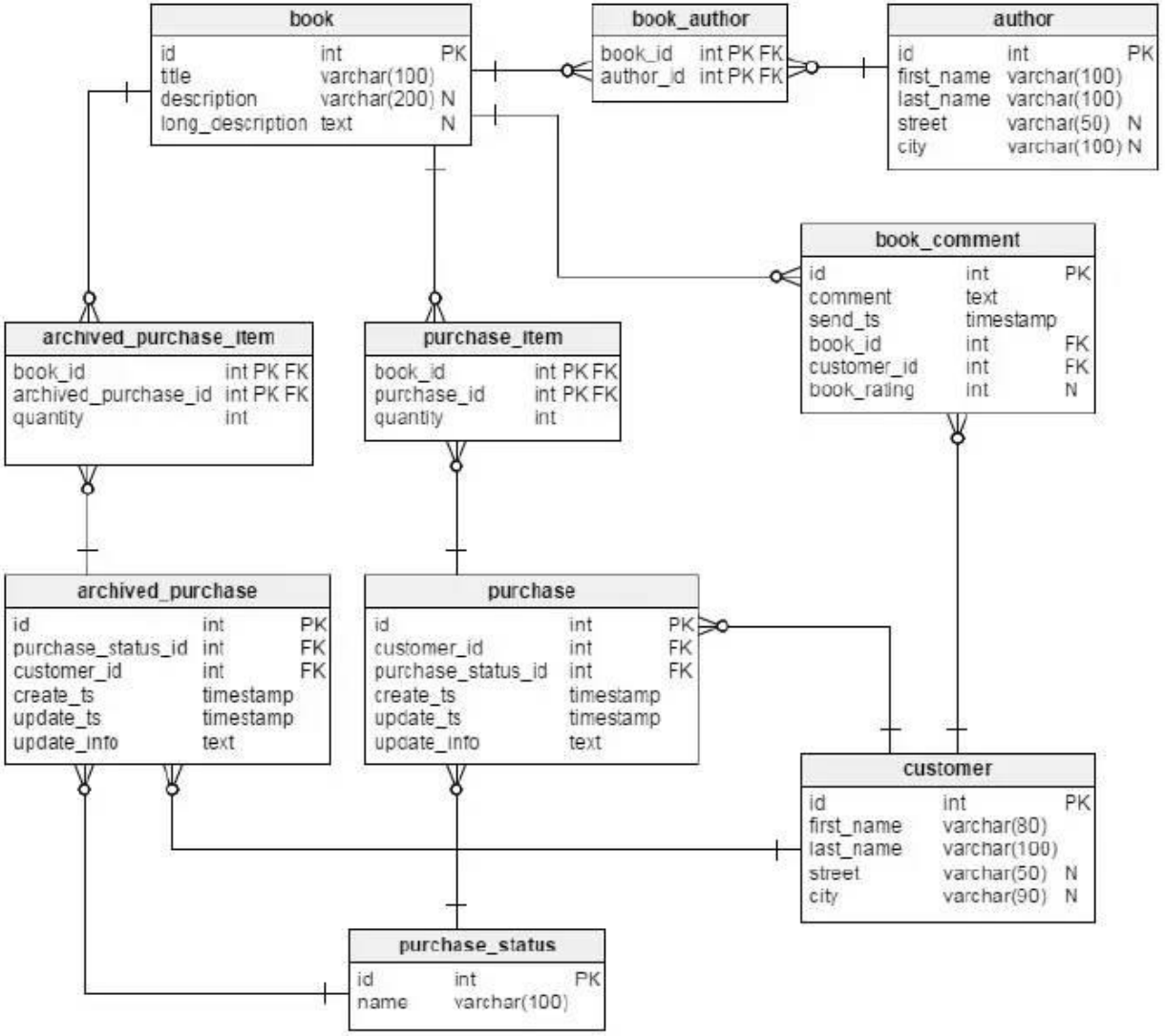
- 管理时区的date和datetime字段算是跨国系统中一个重要的问题
  - 如果你是指自己所在时区的圣诞前夜午夜12点，你必须说“12月24日,23.59 UTC”(即无论你的时区是什么)

- 提示：
  - 检查你的数据库中日期和时间数据类型的细节。SQL Server中Timestamp与PostgreSQL的timestamp完全不同。
  - 用UTC的方式存储日期与时间。
  - 处理好时区问题需要数据库和应用代码直接的合作。确保你理解了数据库驱动的细节。这里有相当多的陷阱。

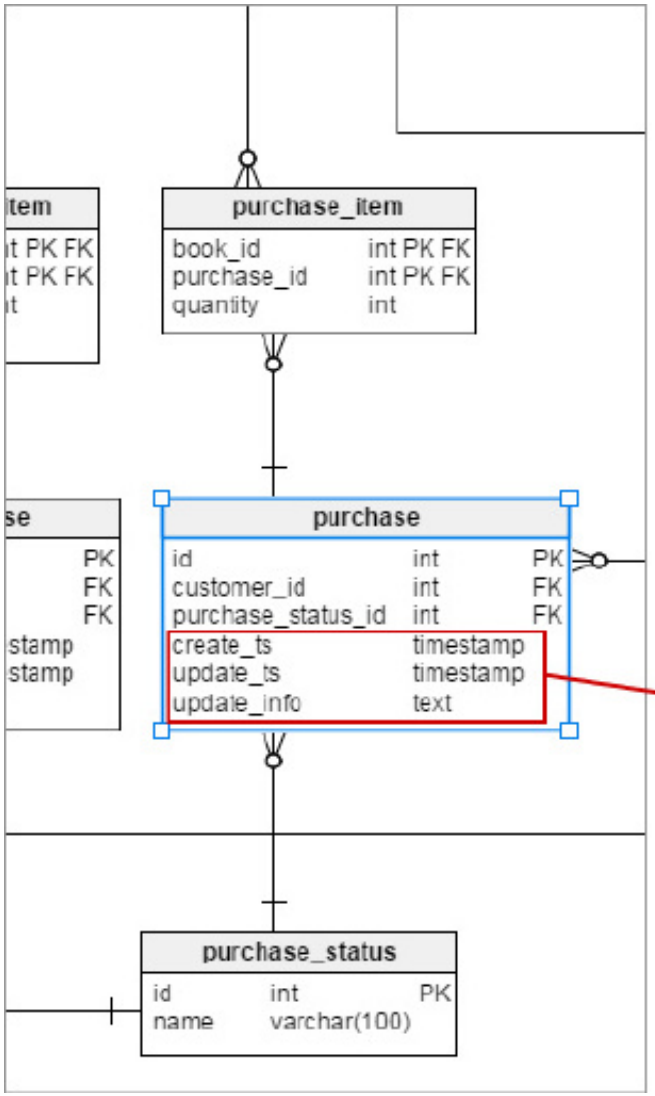
问题6：怎么审计跟踪

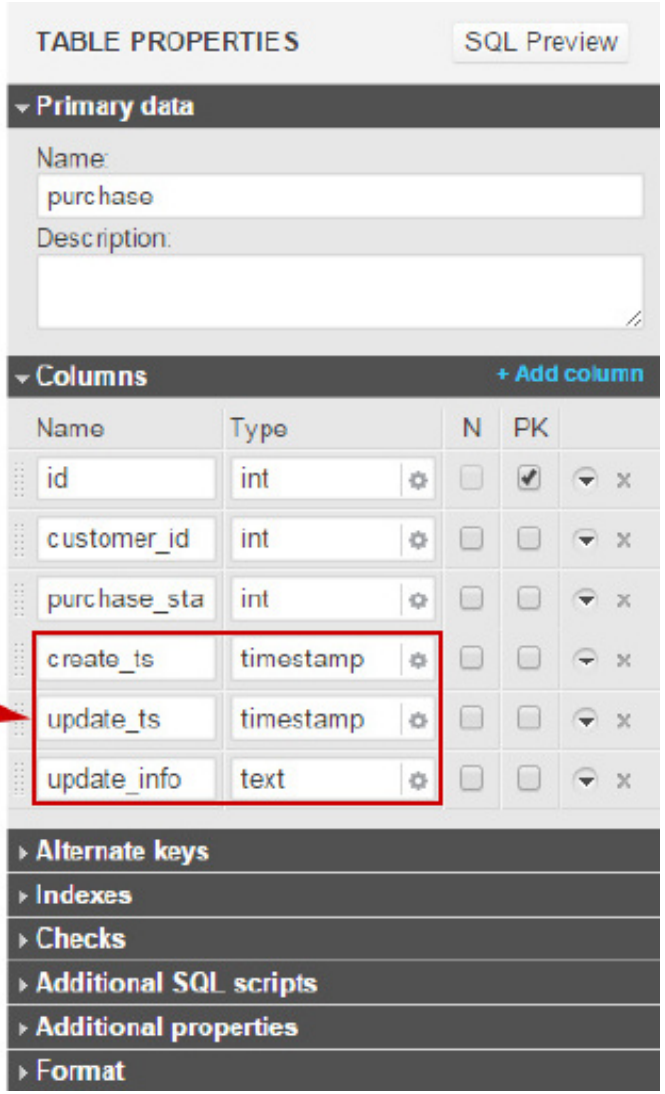
- 数据库中的表可以有创建和更新时间戳，及所创建/修改行的用户标示
- 完整的审计日志可以用触发器或者其它对正在使用的数据库管理系统有效的机制来实现
- 一些审计日志可以存储在单独的数据库以确保无法修改和删除
- 提示：
  - 考虑哪个数据重要到需要跟踪修改/版本化，
  - 考虑风险和成本之间的平衡

模型-0.6


book
id int PK
title varchar(100)
description varchar(200) N
long_description text N
book_author
book_id int PK FK
author_id int PK FK
author
id int PK
first_name varchar(100)
last_name varchar(100)
street varchar(50) N
city varchar(100) N
book_comment
id int PK
comment text
send_ts timestamp
book_id int FK
customer_id int FK
book_rating int N
archived_purchase_item
book_id int PK FK
archived_purchase_id int PK FK
quantity int
purchase_item
book_id int PK FK
purchase_id int PK FK
quantity int
archived_purchase
id int PK
purchase_status_id int FK
customer_id int FK
create_ts timestamp
update_ts timestamp
update_info text
purchase
id int PK
customer_id int FK
purchase_status_id int FK
create_ts timestamp
update_ts timestamp
update_info text
customer
id int PK
first_name varchar(80)
last_name varchar(100)
street varchar(50) N
city varchar(90) N
purchase_status
id int PK
name varchar(100)

模型-0.6-以purchase表为例





问题7：忽略排序规则

- 通常来说，根据字母在字母表中的顺序，我们假定在一种语言中对单词排序与逐字排序一样容易。但是这里有两种陷阱：
  - 首先，哪个字母表？如果我们的内容只有一种语言，那很显然，但是如果内容中有15到30种语言，该由哪一个字母表来决定顺序？
  - 其次，当重音起作用时，逐字排序有时会有错误。

```
db=# select title from book where id between 1 and 4 order
by title collate "POSIX";
title
-------
cote
coté
côte
côté
```

```
db=# select title from book where id between 1 and 4 order
by title collate "en_GB";
title
-------
cote
côte
coté
côté
```

7 common database design errors

The final version of a database model for a small bookstore