一　数据库系统&数据库的开发过程(环节)

- Step1：应用需求分析
  - 功能需求
  - 数据需求

  （编写**数据流程图**＋含数据字典]；**或者**

  采用UML：用例图+时序图+类图等）
- Step2：系统设计
  - 系统功能设计(总统设计，详细设计，界面设计,…)
  - 数据库设计

    概念结构设计*→E-R图

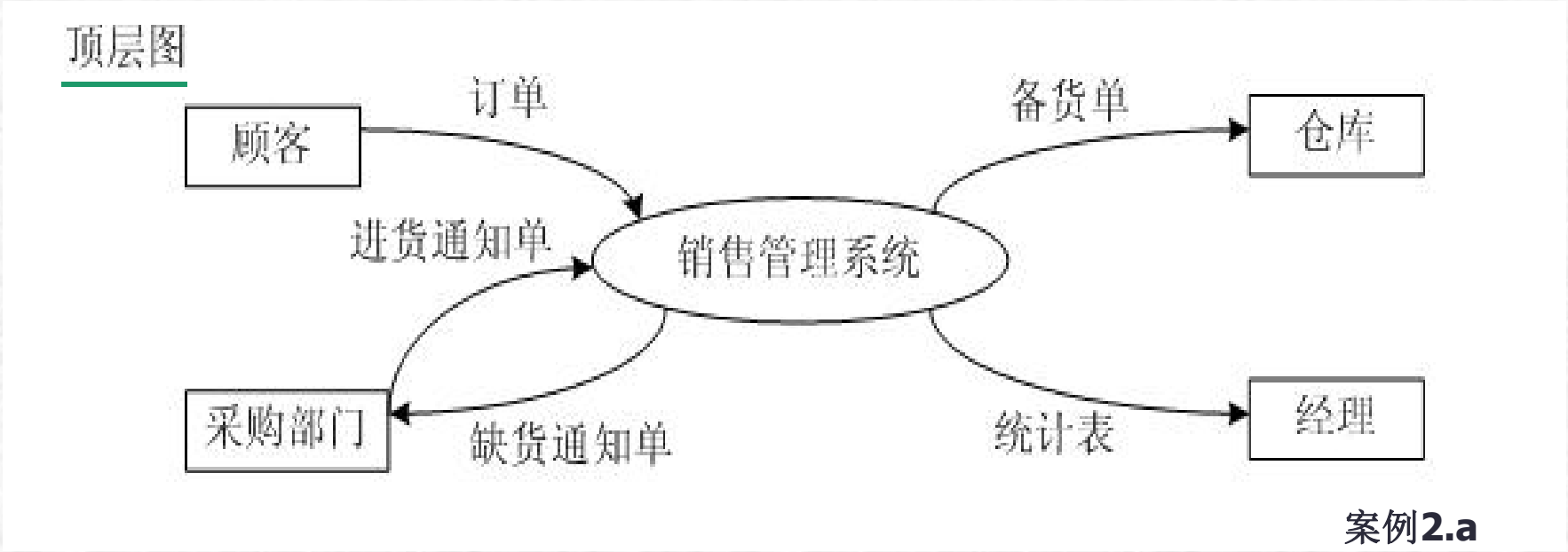
    逻辑结构设计*→关系模式结构，规范化设计

    物理结构设计*→关系模式存放位置＋索引设计
- Step3：系统实现
  - **数据库实现***（实际建立数据结构，SQL实现）
  - 应用功能实现[程序代码,程序检测]

二 数据流程图

1）顶层数据流程图

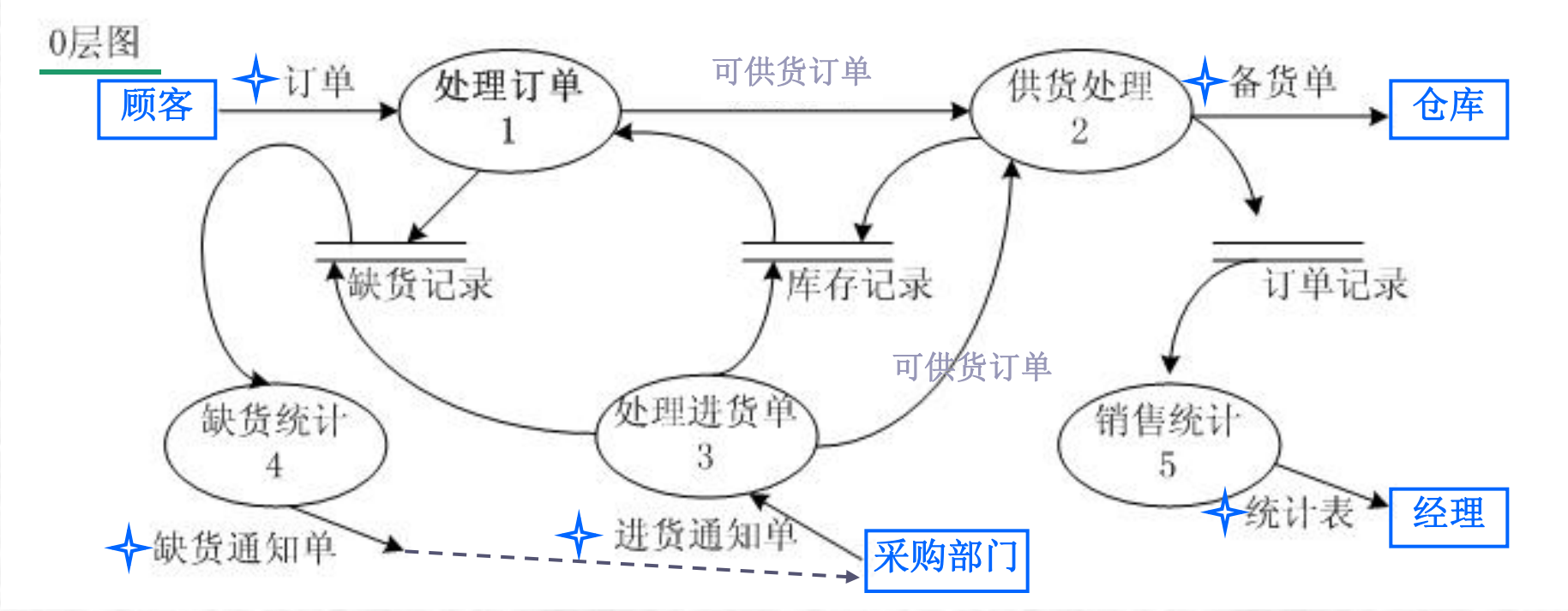
- 用户购买商品、下定单，企业采购商品、保存、供货



案例**2.a**

- 顶层图(输入输出图，仅一张)：把整个系统视为一个加工，并标出系统从外部对象接收哪些数据流和发送哪些数据流到外部对象。

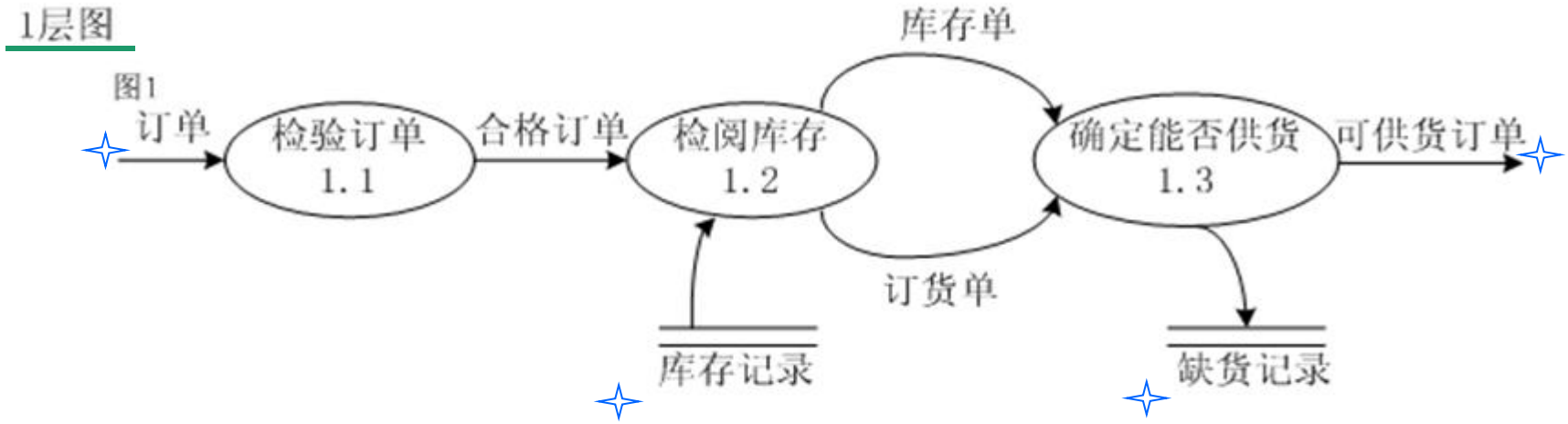
2）0层数据流程图



（2个输入数据流，3个输出数据流）　　案例2.b

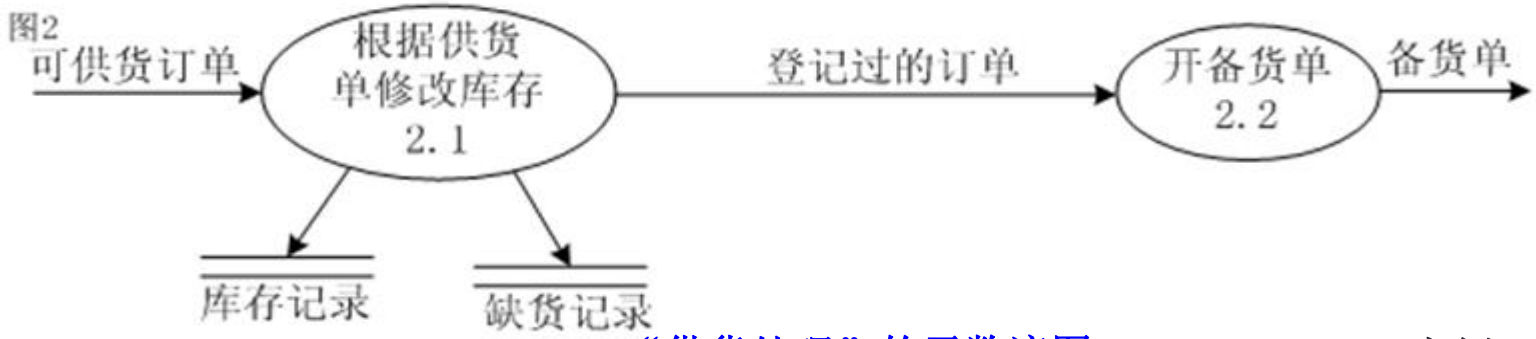
- 0层图(顶层加工细化，仅一张)：对系统的顶层加工进行细化，标出这些加工与外部对象和与这些加工的公共保存文件间的数据流向。

3）分层数据流程图



“处理订单”的子数流图

案例2.c



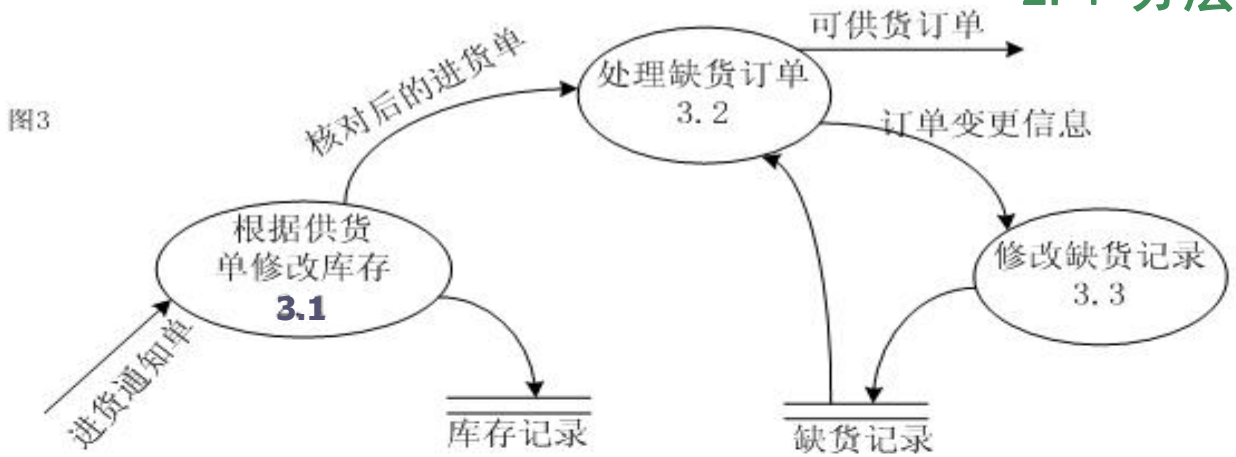
“供货处理”的子数流图

案例2.d

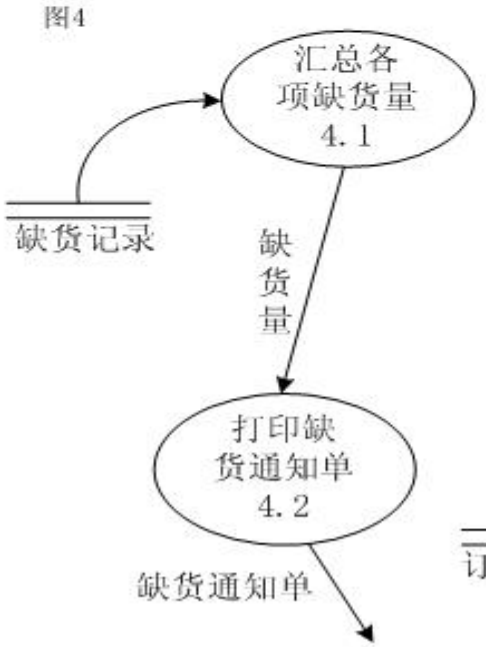
- 画分层数据流图(循环进行)：把上层加工看作是由下层多个子加工形成的子系统，就象从0层图画出1层图一样，画出该加工的分层数据流程图

1层图

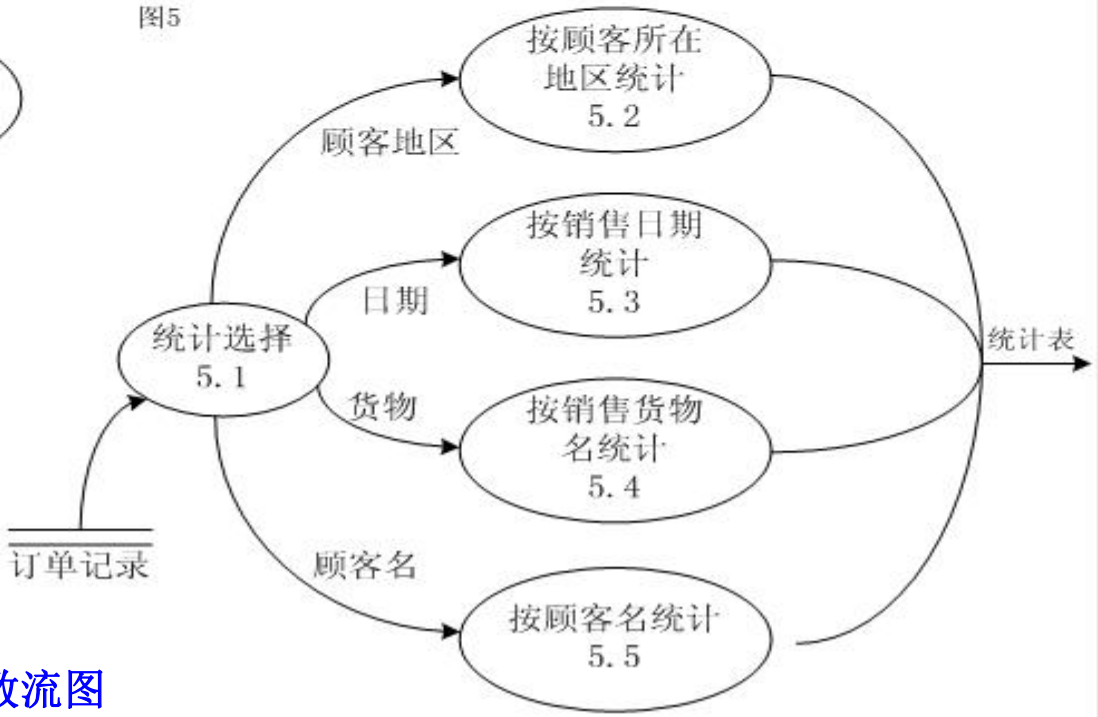
（续）



“处理进货”的子数流图　　案例2.e



案例2.f “缺货统计”的子数流图



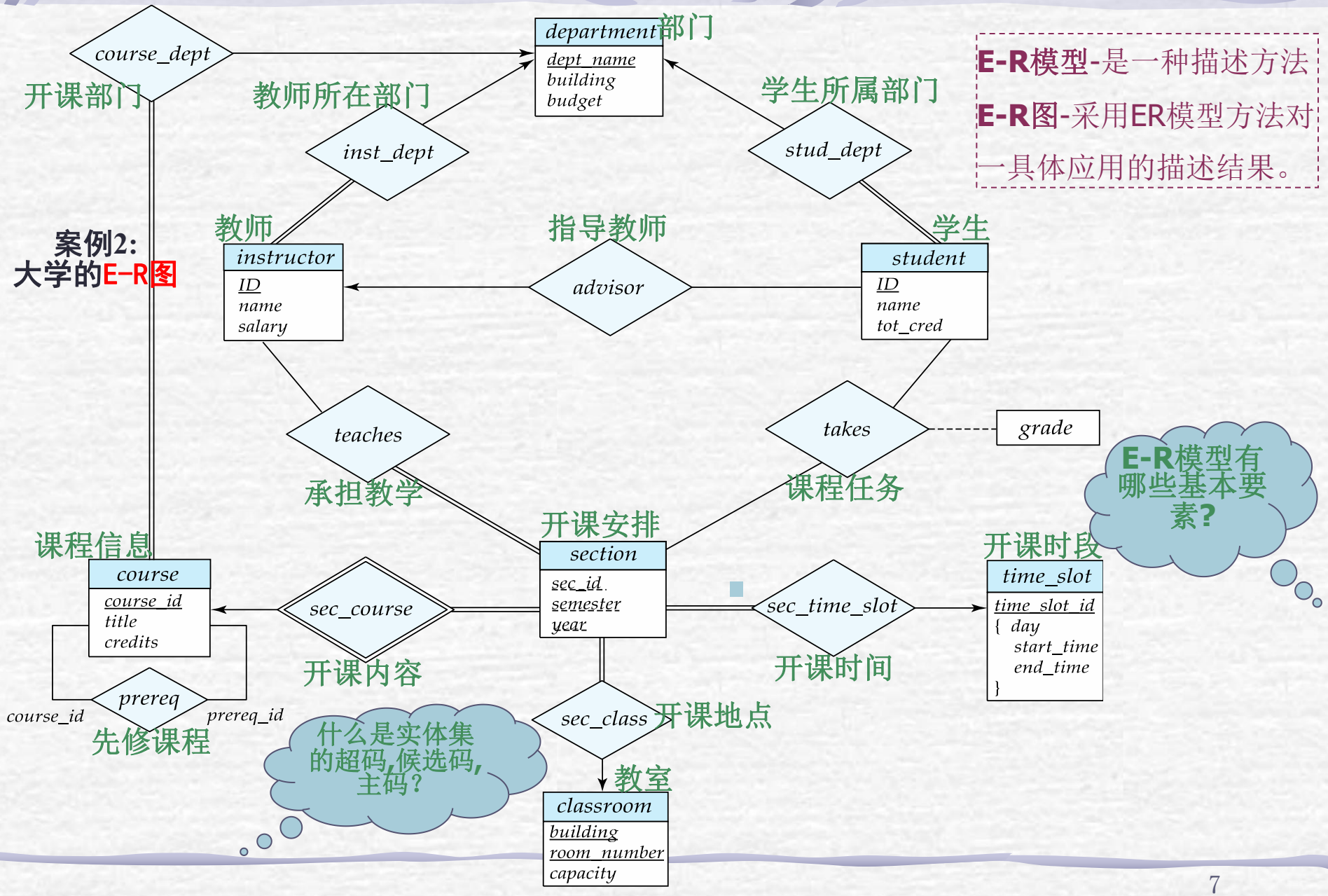
“销售统计”的子数流图　　案例2.g

2.5 数据字典

数据字典包含内容：

- **数据项的详细说明**（是数据的**最小单位**）
- **数据结构的详细说明**（描述某些数据项之间的**组成关系**）
- **数据流的详细说明**（由一个或一组固定的数据项或数据结构**组成结构**）
- **加工的详细说明**（对加工逻辑进行说明）
- **保存文件的详细说明**（描述具体的逻辑存储结构，不涉及物理组织）
- **外部对象的详细说明**（定义外部对象的编号、名称、简述等）

三 实体-联系模型(E-R模型)



E-R模型-是一种描述方法

E-R图-采用ER模型方法对一具体应用的描述结果。

二 查询处理问题

讨论2. 如何基于表达式树进行查询优化？

1. SQL查询语句优化问题

(P.326) 12.2 题

写出一个与此等价的、高效的关系代数表达式，并论证你的选择。

1) 写出SQL查询语句的关系代数表达式?

```
select T.branch_name
from branch T, branch S
where T.assets>S.assets and S.branch_city="Brooklyn"
```

优化提示：

（关系代数表达式的等价变化）

查询优化一般规则：

尽量减少中间结果

选择连接次序

尽早执行选择和投影运算

（可先抽学生小组上来讲解处理基本思路）

（解答见下页）

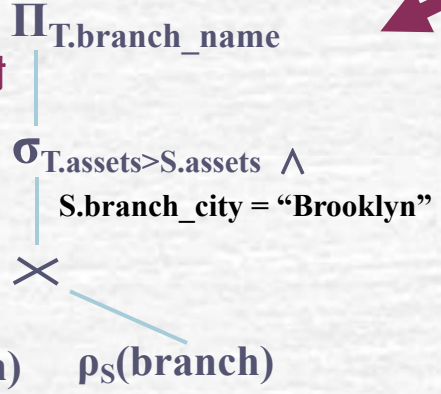
首先易知，上述SQL查询语句对应的关系代数表达式为：

$$\Pi_{T.branch\_name}\ (\sigma_{T.assets>S.assets\ \wedge\ S.branch\_city = \text{“Brooklyn”}}\ (\rho_T(branch)\ X\ \rho_S(branch)))$$

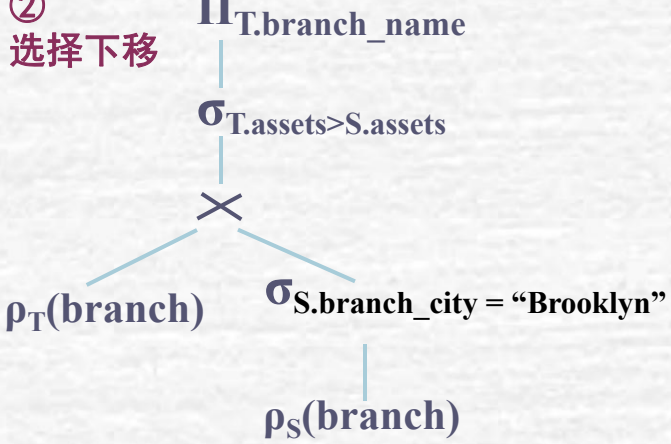
2. SQL查询语句优化方法

$\Pi_{T.branch\_name}\ (\sigma_{T.assets>S.assets\ \wedge\ S.branch\_city = \text{“Brooklyn”}}\ (\rho_T(branch)\ X\ \rho_S(branch))\ )$

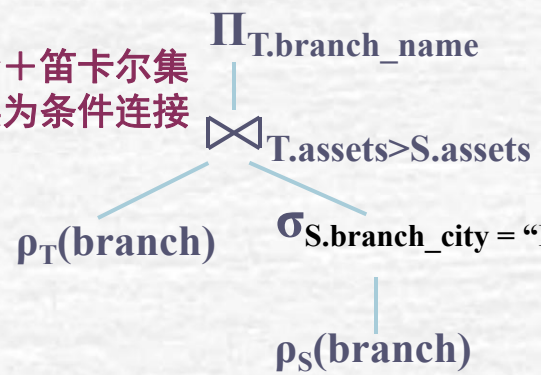
① 初始查询树



② 选择下移

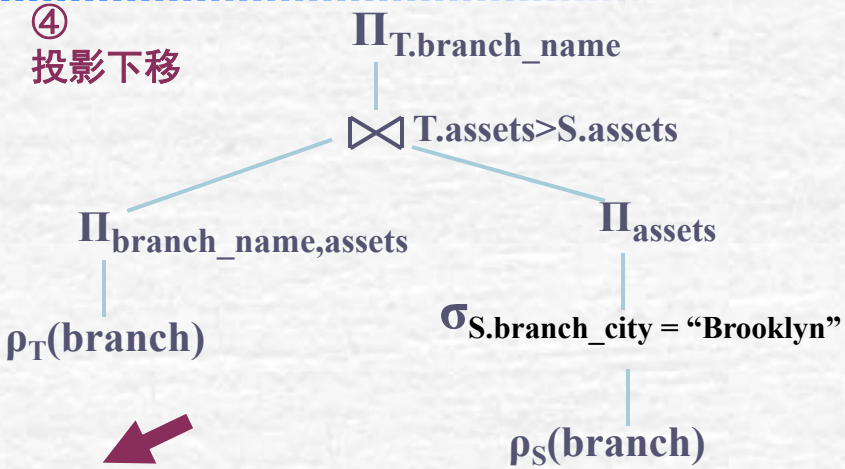


③ 选择＋笛卡尔集转换为条件连接



2) 如何利用启发式规则进行查询优化?

④ 投影下移



⑤ 得到所求关系代数表达式为：

$$\Pi_{T.branch\_name}((\Pi_{branch\_name,\ assets}(\rho_T(branch)))\ \bowtie_{T.assets\ >\ S.assets}\ (\Pi_{assets}\ (\sigma_{(branch\_city\ =\ 'Brooklyn')}(\rho_S(branch)))))$$

三 事务处理问题讨论

(P. 371) 14. 6题

考虑如下图14-16所示的优先图，相应的(并发)调度是冲突可串行化的吗？(解释你的回答)

(若你认为是，请给出一个等价的串行化调度)

讨论3. 优先图在调度可串行化中的重要作用？

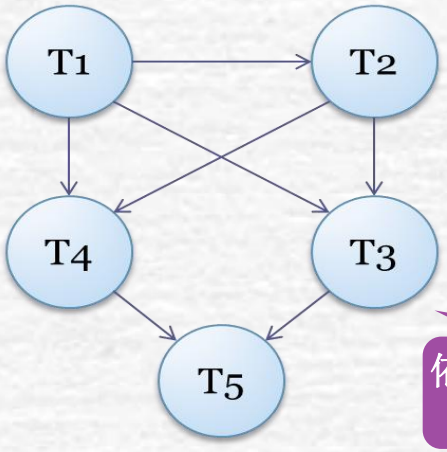
1) 如何利用优先图判定调度可串行化?

2) 如何根据优先图得到其可串行化调度?

(可先抽学生小组回答)

该调度是冲突可串行化的
因优先图中无有向环存在

一个等价的串行化调度：
T1→T2→T3→T4→T5
或 T1→T2→T4→T3→T5



依次去掉“仅有出弧”的事务节点

图14-16 调度优先图

Answer: There is a serializable schedule corresponding to the precedence graph below, since the graph is acyclic. A possible schedule is obtained by doing a topological sort, that is, T_1, T_2, T_3, T_4, T_5.

四 并发控制问题讨论

1. 并发调度控制问题

讨论4. 加锁对并发控制的影响？

1)如何增加所使这两事务符合两段封锁?

2)这两个事务并发执行时会引起死锁吗?

(P.399) 15.2题

考虑下面两个事务：

```
T34:    read(A);
        read(B);
        if A=0 then B:=B+1;
        write(B);

T35:    read(B);
        read(A);
        if B=0 then A:=A+1;
        write(A);
```

a. 给事务T34与T35增加加锁、解锁指令，使它们遵从两阶段封锁协议。

b. 这两个事务会引起死锁吗？

（先抽学生小组回答）

说明：两阶段封锁协议

(保证冲突可串行化，但不保证不发生死锁)

1. 增长阶段：事务可以获得锁，但不能释放锁；
2. 缩减阶段：事务可以释放锁，但不能获得新锁。

2. 并发调度控制问题解答

解答a. 增加加锁、解锁指令后：

T 34:	T 35:
lock-S(A) read(A) lock-X(B) read(B) if A = 0 then B := B + 1 write(B) unlock(A) unlock(B)	lock-S(B) read(B) lock-X(A) read(A) if B = 0 then A := A + 1 write(A) unlock(B) unlock(A)

解答b. 有可能发生死锁。如：

T34	T35
lock-S(A) read(A)	
	lock-S(B) read(B)
lock-X(B) read(B) if A = 0 then B := B + 1 write(B) unlock(A) unlock(B)	
	lock-X(A) read(A) if B = 0 then A := A + 1 write(A) unlock(B) unlock(A)

等待T35释放对资源B的共享锁 →

← 等待T34释放对资源A的共享锁

五 恢复技术问题讨论

1. 日志的扫描方向

讨论5. 日志扫描次序对数据恢复的影响？

（P.427）16.1题

请解释：a) 为什么undo-list中事务的日志记录，必须由后往前进行处理，

b) 而redo-list中事务的日志记录，必须由前往后进行处理。

（可先抽学生小组回答）

1)恢复数据时日志的正确扫描方向?

解答：a）

分析对一个单一的事务做undo。

假设一个数据项在这个事务中被修改了多次，首先从1修改为2，又从2修改为3；

如果undo-list的日记记录从前往后处理，那么最后这个数据项的值会被恢复为2；

而从后往前处理，结果是1，这才是希望恢复到的结果。

对多个事务进行undo，可同理分析。

解答：b）

对上同样的例子分析。

假设事务已提交，对其进行redo。若从后往前执行，结果是2（不正确）；

若从前往后执行，结果是3（正确）。

2. 延迟修改技术与日志

(P.426) 16.5 a.题

假设在数据库中使用延迟修改技术。

a.更新日志记录中的旧值还需要吗？为什么？

2)采用延迟修改技术还需记录旧值吗?

（可先抽学生小组回答）

说明：延迟修改技术(p.407)

如果一个事务直到它提交时都没有修改数据库(没有执行对磁盘缓冲区获磁盘自身的修改)，就称采用了延迟修改技术.

(即：将一个事务的所有write操作拖延到事务部分提交时才执行，来保证事务的原子性)。

解答a.

不需要！理由如下：

1) 如果事务已经提交，那么旧值就不再需要了，因为已提交的事务不再进行undo操作；

2) 如果事务没有提交，在执行过程中遇到系统崩溃，那么数据库还没有被修改，不需要利用旧值对数据库进行恢复。