

2017年数据库试卷 中英文问题 +答案+解析

数据库填空题 中英文问题+答案 +解析

1.

英文问题

Given a superkey K of a relation schema R , if no proper subset of K is a superkey, we call this K as a __ of R .

中文问题

给定关系模式 R 的超键 K ，若 K 的任意真子集都不是超键，则称 K 是 R 的__。

答案

candidate key (候选码)

解析

- **超键**：能唯一标识关系元组的属性集；
- **候选码**：是“最小的超键”——去掉任意一个属性后就不再是超键，因此候选码是超键的“最小形式”。

2.

英文问题

In SQL, if we want to delete the attributes of a schema, we use the instruction __; while if we want to delete tuples of a relation, we use the instruction __.

中文问题

在SQL中，若要删除模式的属性，使用的语句是__；若要删除关系的元组，使用的语句是__。

答案

ALTER TABLE DROP COLUMN; DELETE

解析

- **删除属性**：需修改表结构，用 ALTER TABLE 语句的 DROP COLUMN 子句（如 ALTER TABLE student DROP COLUMN age;）；
- **删除元组**：是操作表中的数据，用 DELETE 语句（如 DELETE FROM student WHERE id='001';）。

3.

英文问题

SQL supports three-valued logic: true, false and unknown. Please indicate the result of the operation: unknown OR false is __.

中文问题

SQL支持三值逻辑：true（真）、false（假）、unknown（未知）。请指出运算 unknown OR false 的结果是__。

答案

unknown

解析

三值逻辑的OR运算规则：

- true OR 任意值 → true；
- false OR unknown → unknown（只有当其中一个值为true时，结果才为true；否则若有unknown，结果为unknown）。

4.

英文问题

In E-R model, if some entities in E may not participate in any relationship in R , the participation of entity set E in relationship R is said to be __.

中文问题

在E-R模型中，若实体集 E 中的部分实体可能不参与关系 R ，则称实体集 E 在关系 R 中的参与类型是__。

答案

partial participation（部分参与）

解析

E-R模型中实体集的参与类型：

- **部分参与**：实体集的部分实体可以不参与关系（用**虚线**表示）；
- **完全参与**：实体集的所有实体都必须参与关系（用**实线**表示）。

5.

英文问题

Normally an entity of a E-R diagram is mapped into one relation schema. However, if an entity contains a __ attribute, the entity may possibly be mapped into multiple schemas.

中文问题

通常E-R图中的一个实体映射为一个关系模式。但如果实体包含__属性，则该实体可能被映射为多个关系模式。

答案

composite（复合）/multivalued（多值）

解析

- **复合属性**：属性可再分（如“地址”包含“街道、城市”），需拆分为多个属性，可能拆分到不同关系；
- **多值属性**：一个实体的该属性对应多个值（如“电话”可能有多个号码），需单独用一个关系存储（避免元组重复），因此实体需映射为多个关系。

6.

英文问题

BCNF decomposition can only guarantee it is lossless, while 3NF decomposition can guarantee that it is both lossless and __.

中文问题

BCNF分解只能保证无损连接，而3NF分解可以保证既无损连接又__。

答案

dependency-preserving（保持依赖）

解析

- **BCNF分解**：仅保证“无损连接”，但可能丢失原函数依赖；
- **3NF分解**：能同时保证“无损连接”和“保持依赖”（原函数依赖可由分解后的关系推导）。

7.

英文问题

In a B+-tree index, if a non-leaf node contains at most n pointers, then the leaf node have at least __ index entries.

中文问题

在B+树索引中，若非叶节点最多包含 n 个指针，则叶节点至少包含__个索引项。

答案

$\lceil n/2 \rceil$ ($n/2$ 向上取整)

解析

B+树的阶数定义：非叶节点最多有 n 个指针（对应 $n - 1$ 个索引项）。

B+树的“最小填充率”为50%：

- 叶节点的索引项数量范围是 $\lceil n/2 \rceil \sim n - 1$ ，因此至少有 $\lceil n/2 \rceil$ 个索引项。

8.

英文问题

There are two approaches to implement a SELECT operation in database query processing, i.e., file scanning, and __.

中文问题

数据库查询处理中实现SELECT操作的两种方法是：文件扫描和__。

答案

indexing（索引扫描）

解析

实现SELECT（选择）操作的两种核心方法：

- **文件扫描**：全表遍历，适用于无索引的情况；
- **索引扫描**：利用索引快速定位符合条件的元组，效率更高。

Part I 填空题（每题2分，共20分）

9. 事务状态

英文问题

There are five states for a transaction: active, partially committed, failed, aborted, and committed. A transaction often starts from an active state, and ends at a committed or __ state.

中文问题

事务有五种状态：活动状态（active）、部分提交状态（partially committed）、失败状态（failed）、中止状态（aborted）和提交状态（committed）。事务通常从活动状态开始，最终结束于提交状态或_状态。

答案

aborted（中止）

解析

事务的生命周期：从“活动状态”开始执行操作，若执行中出现错误进入“失败状态”，进而转为“中止状态”；若顺利执行完所有操作进入“部分提交状态”，最终完成持久化后进入“提交状态”。因此事务的最终状态只能是“提交”或“中止”。

10. 两阶段锁协议

英文问题

Although the two-phase locking protocol cannot avoid __, it can guarantee that all the possibly generated schedules under the protocol are serializable.

中文问题

尽管两阶段锁协议无法避免__，但它能保证在该协议下生成的所有调度都是可串行化的。

答案

dead lock（死锁）

解析

两阶段锁协议（2PL）的核心作用是保证冲突可串行化，但它允许事务在“增长阶段”持续获取锁，可能出现“循环等待”（如T1持有A锁等待B锁，T2持有B锁等待A锁），从而引发死锁，因此无法避免死锁。

Part II 术语解释与简答题（每题6分，共24分）

1. 物化视图（View Materialization）与SQL视图定义

(a) 解释物化视图（View Materialization）

英文问题

Explain what is view materialization.

中文问题

解释什么是物化视图（View Materialization）。

答案

A materialized view is a special database object that stores the precomputed result of a query persistently. Unlike a normal virtual view (which only stores the SQL definition and computes data dynamically when queried), a materialized view directly reads data from its own stored result when queried. When the data of the base tables (tables involved in the query) changes, the materialized view needs to be refreshed (manually or automatically) to maintain data consistency.

（物化视图是一种特殊的数据库对象，它持久化存储查询的预计算结果。与普通虚拟视图（仅存储SQL定义，查询时动态计算数据）不同，查询物化视图时直接从其存储的结果中读取数据。当基表（查询涉及的表）数据发生变化时，需刷新（手动或自动）物化视图以保证数据一致性。）

解析

物化视图的核心是“空间换时间”：通过预先存储复杂查询（如多表连接、聚合操作）的结果，大幅提升后续查询效率，适用于数据仓库、复杂报表等场景；但需额外存储资源，且需通过“刷新机制”解决基表数据更新导致的物化视图数据过期问题。

(b) 用SQL定义视图（统计各公司员工数）

英文问题

Define a view in SQL (named `num_employee`, with columns `company_name` and `num`) that gives the number of employees in each

company. The related tables are `employee(employee_name, street, city)`, `works(employee_name, company_name, salary)`, `company(company_name, city)`.

中文问题

用SQL定义一个名为`num_employee`的视图，包含`company_name`（公司名称）和`num`（员工数量）两列，用于统计每个公司的员工数量。相关表结构为`employee(employee_name, street, city)`（员工表）、`works(employee_name, company_name, salary)`（工作表）、`company(company_name, city)`（公司表）。

答案

```
CREATE VIEW num_employee(company_name, num)
AS SELECT company_name, COUNT(DISTINCT employee_name)
FROM works
GROUP BY company_name;
```

解析

- 统计员工数需基于`works`表（记录员工与公司的关联关系），无需关联`employee`或`company`表（`works`已包含`employee_name`和`company_name`）；
- 使用`COUNT(DISTINCT employee_name)`避免同一员工在同一公司的多条记录重复计数；
- `GROUP BY company_name`按公司分组，确保统计每个公司的独立员工数量。

2. 顺序索引（Sequential Index）与哈希索引（Hash Index）比较

(a) 解释顺序索引（Sequential Index）

英文问题

Explain what is a sequential index.

中文问题

解释什么是顺序索引（Sequential Index）。

答案

A sequential index is an index structure where search keys (the attributes the index is built on) are stored in sorted order. The index entries (each consisting of a search key and a pointer to the corresponding record) are arranged in the same order as the search key values. Sequential indexes are closely related to the physical storage order of the base table—if the base table’s records are sorted by the search key, the sequential index is called a clustered (primary) index; otherwise, it is a non-clustered (secondary) index.

（顺序索引是一种索引结构，其搜索键（建立索引的属性）按排序顺序存储。索引项（每个索引项包含一个搜索键和指向对应记录的指针）的排列顺序与搜索键值的顺序一致。顺序索引与基表的物理存储顺序密切相关：若基表记录按搜索键排序，则该顺序索引称为聚簇（主）索引；否则称为非聚簇（二级）索引。）

解析

顺序索引的核心是“有序性”，类比书籍的目录（按章节号/标题排序），通过有序的索引项快速定位记录，支持范围查询（如“查询薪资3000-5000的员工”），是数据库中最常用的索引类型之一。

(b) 比较顺序索引与哈希索引，并描述顺序索引的优缺点

英文问题

Compare sequential index with hash index, and describe the advantages and disadvantages of sequential index.

中文问题

比较顺序索引与哈希索引，并描述顺序索引的优缺点。

答案

Comparison Dimension	Sequential Index	Hash Index
Applicable Query Types	Supports both equality and range queries	Only supports equality queries
Cost of Equality Query	$O(\log n)$ (traverses the sorted index tree)	$O(1)$ (direct hash mapping, no collisions)

Comparison Dimension	Sequential Index	Hash Index
Cost of Range Query	Low (reads sequentially from start to end)	High (needs to scan all hash buckets)
Data Order Dependency	Relies on sorted search key values	Relies on hash function (no order)

Advantages of Sequential Index:

- 1. Supports range queries (e.g., `salary > 5000 AND salary < 8000`), which hash indexes cannot handle efficiently.
- 2. Works well with sorted base tables, reducing the cost of record retrieval.
- 3. Can be optimized for both read and write operations (e.g., B+ trees as sequential indexes balance insert/delete costs).

Disadvantages of Sequential Index:

- 1. Equality query cost is higher than hash indexes (needs to traverse the index tree instead of direct mapping).
- 2. Insert/delete operations may require adjusting the index structure (e.g., B+ tree splitting/merging), increasing overhead.
- 3. Performance degrades if the base table is frequently updated and becomes unsorted (for non-clustered sequential indexes).

(顺序索引与哈希索引比较：

比较维度	顺序索引	哈希索引
适用查询类型	支持等值查询和范围查询	仅支持等值查询
等值查询代价	$O(\log n)$ (遍历有序索引树)	$O(1)$ (无冲突时直接哈希映射)
范围查询代价	低 (从起始位置顺序读取至结束)	高 (需扫描所有哈希桶)
数据顺序依赖	依赖搜索键值的有序性	依赖哈希函数 (无顺序)

顺序索引的优点：

1. 支持范围查询（如“salary > 5000 AND salary < 8000”），哈希索引无法高效处理此类查询；
2. 与有序基表配合良好，降低记录检索代价；
3. 可兼顾读写性能（如B+树作为顺序索引，平衡插入/删除开销）。

顺序索引的缺点：

1. 等值查询代价高于哈希索引（需遍历索引树，而非直接映射）；
2. 插入/删除操作可能需调整索引结构（如B+树的分裂/合并），增加开销；
3. 若基表频繁更新导致无序（非聚簇顺序索引），性能会下降。）

解析

顺序索引的核心优势在于“有序性”，使其能高效处理范围查询；而哈希索引的核心优势是“直接映射”，等值查询速度更快。实际应用中，若需频繁执行范围查询，优先选择顺序索引；若仅需等值查询（如按主键精确查找），可选择哈希索引。

3. 块嵌套循环连接 (Nested Block Loop Join)

(a) 解释块嵌套循环连接算法的主要步骤

英文问题

Nested block loop join algorithm is an effective algorithm to implement the JOIN operation in database processing. Please explain the main steps of the algorithm.

中文问题

块嵌套循环连接算法是数据库处理中实现JOIN操作的有效算法，请解释该算法的主要步骤。

答案

The block nested loop join algorithm optimizes the simple nested loop join by using memory buffers to reduce disk I/O. Its main steps are as follows:

1. **Buffer Allocation:** Allocate a memory buffer for the outer relation (e.g., relation R). The buffer can store multiple blocks of R (instead of a single tuple in the simple nested loop join).
2. **Outer Relation Scanning:** Read blocks of the outer relation R into the buffer one by one. For each batch of blocks in the buffer:

3. **Inner Relation Scanning:** Read all blocks of the inner relation (e.g., relation `S`) one by one. For each block of `S`:
4. **Tuple Matching:** For each tuple in the buffered blocks of `R`, compare it with each tuple in the current block of `S`. If the tuple pair satisfies the join condition, add the combined result to the final output.
3. **Termination:** Repeat steps 2-4 until all blocks of the outer relation `R` are processed.

(块嵌套循环连接算法通过内存缓冲区减少磁盘I/O，优化了简单嵌套循环连接，主要步骤如下：

1. **缓冲区分配:** 为外关系（如关系`R`）分配一块内存缓冲区，可存储`R`的多个块（而非简单嵌套循环连接中的单个元组）；
2. **扫描外关系:** 将外关系`R`的块逐一读入缓冲区，对于缓冲区中的每一批块：
3. **扫描内关系:** 将内关系（如关系`S`）的所有块逐一读取，对于`S`的每个块：
4. **元组匹配:** 将缓冲区中`R`的每个元组与`S`当前块的每个元组进行比较，若满足连接条件，则将合并后的结果加入最终输出；
3. **终止:** 重复步骤2-4，直至外关系`R`的所有块处理完毕。)

解析

块嵌套循环连接的核心优化是“以块为单位处理外关系”：相比简单嵌套循环（每次处理1个外关系元组，需多次读取内关系），块级处理可减少内关系的读取次数（如缓冲区存10个外关系块，内关系只需读1次即可匹配10个块的元组），大幅降低磁盘I/O开销。

(b) 估算块嵌套循环连接的代价 (`instructor ⋈ teaches`)

英文问题

Estimate the cost of the relational algebra `instructor ⋈ teaches` by using the nested block loop join algorithm. Suppose the relation `instructor` has 1000 blocks, the relation `teaches` has 500 blocks, the main memory has 5 data blocks. The cost is measured in number of block transfers and disk seeks.

中文问题

使用块嵌套循环连接算法估算关系代数 $\text{instructor} \bowtie \text{teaches}$ (教师表与授课表连接) 的代价, 代价以磁盘块传输数和寻道数衡量。已知:

instructor 表有1000个块, teaches 表有500个块, 主内存有5个数据块。

答案

1. Number of Block Transfers (块传输数)

The formula for block transfers in block nested loop join is:

$$\text{Block transfers} = (\text{Number of blocks of outer relation}) + (\text{Number of batches of outer relation}) \times (\text{Number of blocks of inner relation})$$

- Choose instructor as the outer relation (more blocks, reducing inner relation scans).
- Number of batches of outer relation: $\text{ceil}(\text{Blocks of outer relation} / (\text{Memory blocks} - 2)) = \text{ceil}(1000 / (5 - 2)) = \text{ceil}(1000/3) \approx 334$ (reserve 2 blocks for inner relation and result).
- Block transfers = $1000 + 334 \times 500 = 1000 + 167000 = 168000$

2. Number of Disk Seeks (寻道数)

Each batch of outer relation requires 1 seek (to read the outer block batch) + 1 seek (to read all inner blocks).

- Number of seeks = $\text{Number of batches} \times 2 = 334 \times 2 = 668$

(1. 块传输数

块嵌套循环连接的块传输数公式:

$$\text{块传输数} = \text{外关系块数} + \text{外关系批次数} \times \text{内关系块数}$$

- 选择 instructor 作为外关系 (块数更多, 减少内关系扫描次数) ;
- 外关系批次数: $\text{ceil}(\text{外关系块数} / (\text{内存块数} - 2)) = \text{ceil}(1000 / (5 - 2)) = \text{ceil}(1000/3) \approx 334$ (预留2个块给内关系和结果存储) ;
- 块传输数 = $1000 + 334 \times 500 = 1000 + 167000 = 168000$

2. 寻道数

每批外关系处理需1次寻道 (读取外关系块批) + 1次寻道 (读取所有内关系块) :

- 寻道数 = $\text{批次数} \times 2 = 334 \times 2 = 668$

解析

- 内存块分配：需预留2个块（1个用于读取内关系块，1个用于存储连接结果），因此外关系可使用的缓冲区块数为 $5 - 2 = 3$ ；
- 批次数计算：外关系总块数除以缓冲区块数，向上取整（不足1批按1批算）；
- 寻道数逻辑：每批外关系只需1次寻道定位外块，内关系所有块可通过1次寻道连续读取（假设内关系块物理连续），因此每批对应2次寻道。

4. 可串行化调度 (Serializable Schedule)

(a) 解释可串行化调度

英文问题

Explain what is a serializable schedule.

中文问题

解释什么是可串行化调度 (Serializable Schedule) 。

答案

A schedule (order of transaction operations) is serializable if its execution result is equivalent to the result of executing the transactions in some serial order (i.e., no interleaving of operations between transactions). There are two main types of serializability:

1. **Conflict Serializability:** A schedule is conflict-serializable if it can be transformed into a serial schedule by swapping non-conflicting operations (e.g., `T1.Read(A)` and `T2.Read(B)`). It can be checked by verifying that the precedence graph (with transactions as nodes and conflicting operations as directed edges) has no cycles.
2. **View Serializability:** A schedule is view-serializable if it is view-equivalent to a serial schedule (i.e., same initial reads, same read-from dependencies, and same final writes for all data items). It is a broader concept than conflict serializability but has no efficient checking algorithm in practice.

（可串行化调度是指：调度（事务操作的执行顺序）的执行结果，与按某一串行顺序（事务间无操作交错）执行这些事务的结果完全等价。可串行化主要分为两类：

1. **冲突可串行化**：若调度可通过交换非冲突操作（如 $T1.Read(A)$ 和 $T2.Read(B)$ ）转换为串行调度，则为冲突可串行化。可通过“优先图（事务为节点，冲突操作为有向边）无环”验证。
2. **视图可串行化**：若调度与某串行调度“视图等价”（即所有数据项的初始读、读依赖、最终写均相同），则为视图可串行化。它是比冲突可串行化更宽泛的概念，但实际中无高效的验证算法。）

解析

可串行化是数据库并发控制的核心目标：既允许事务并发执行以提高效率，又保证结果与串行执行一致（避免数据不一致）。实际数据库系统中，多采用“冲突可串行化”（通过两阶段锁协议等实现），因其验证和实现更高效。

(b) 判断调度是否可串行化

英文问题

Detect if the following schedule of two transactions is serializable.

Please justify your answer.

The schedule ($T1: Read(B), \text{If } B=0 \text{ then } A=A+1, Write(A); T2: Read(A), Read(B), \text{If } A=0 \text{ then } B=B+1, Write(B)$) with interleaved operations as follows:

$T2.Read(A) \rightarrow T2.Read(B) \rightarrow T2.If\ A=0\ then\ B=B+1 \rightarrow T1.Read(B) \rightarrow T1.If\ B=0\ then\ A=A+1 \rightarrow T2.Write(B) \rightarrow T1.Write(A)$

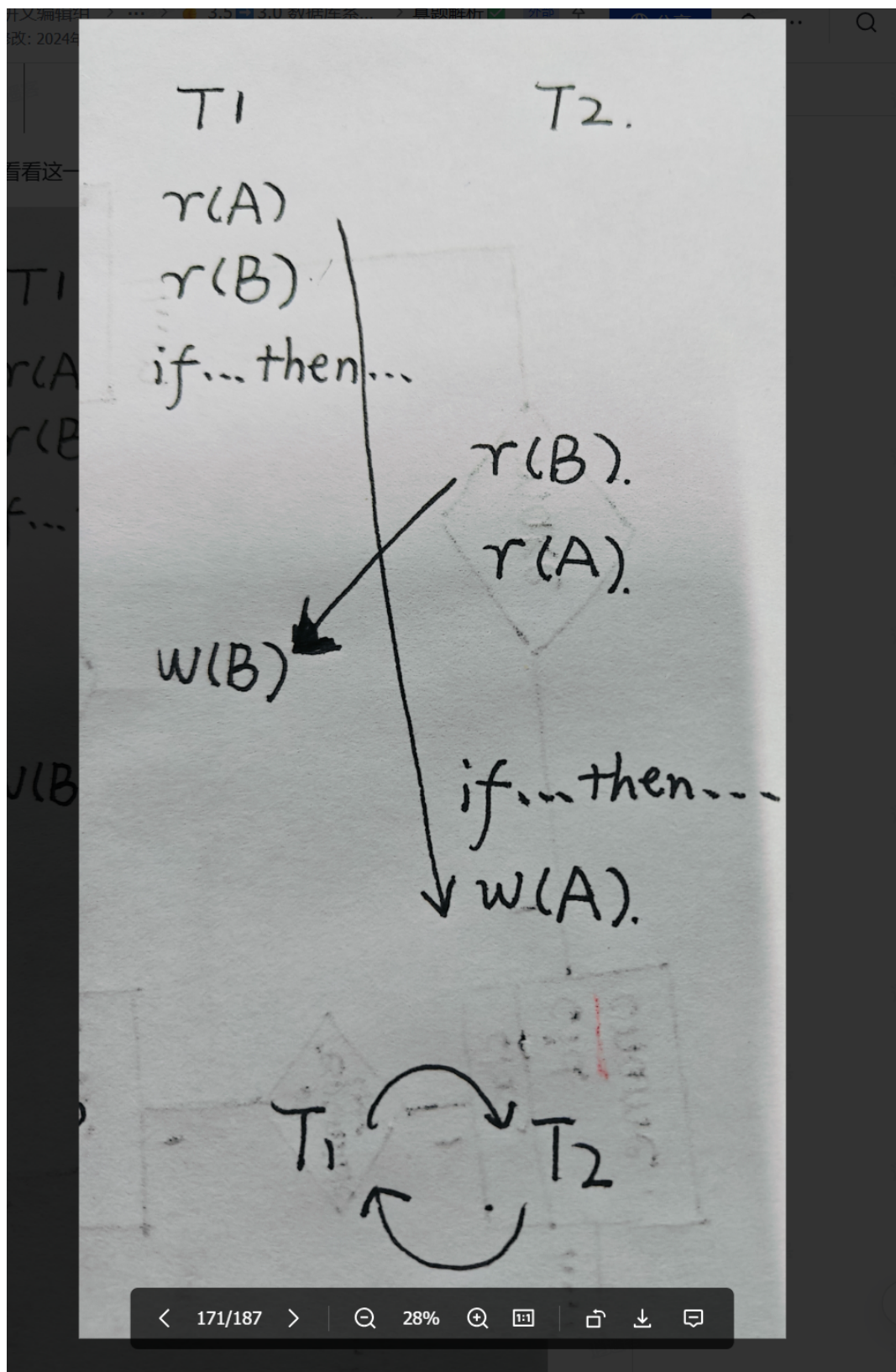
中文问题

判断以下两个事务的调度是否可串行化，并说明理由。

事务 $T1$ ：Read(B)（读B）、If $B=0$ then $A=A+1$ （若 $B=0$ 则 A 加 1）、Write(A)（写 A）；

事务 $T2$ ：Read(A)（读A）、Read(B)（读B）、If $A=0$ then $B=B+1$ （若 $A=0$ 则 B 加 1）、Write(B)（写B）；

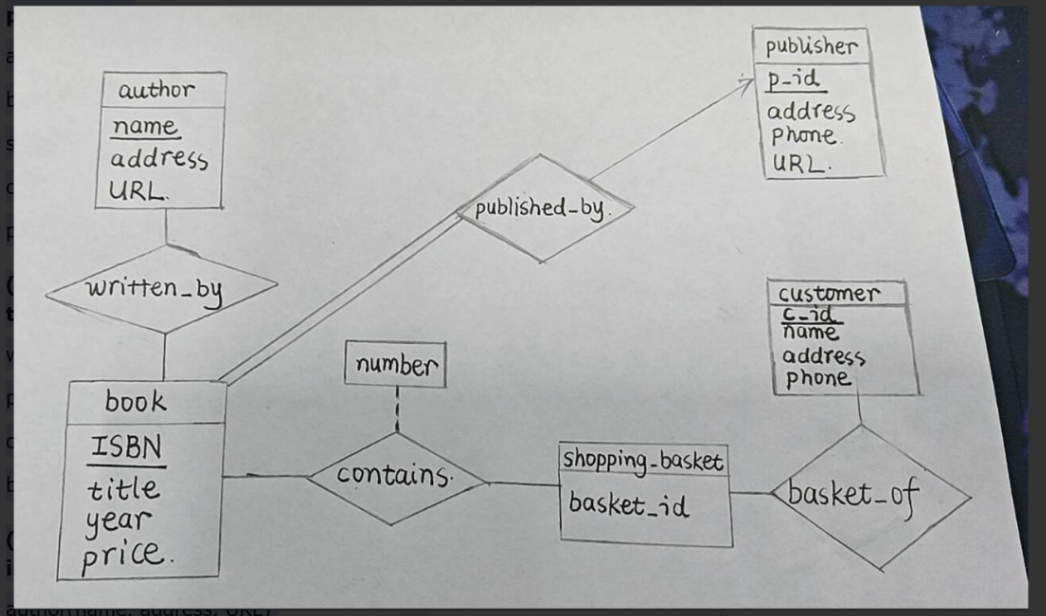
调度的操作 interleaving 顺序： $T2.Read(A) \rightarrow T2.Read(B) \rightarrow T2.If\ A=0\ then\ B=B+1 \rightarrow T1.Read(B) \rightarrow T1.If\ B=0\ then\ A=A+1 \rightarrow T2.Write(B) \rightarrow T1.Write(A)$



问题

1. 请给出实体集的模式，并下划线标注主键。
2. 请给出关系集的模式，并下划线标注主键。
3. 去除上述模式的冗余，即必要时合并一些模式。

(a) Give the schemas of the entity sets, and underline the



答案

(a) 实体集模式:

- `author(name, address, URL)`, 主键为 `name`
- `book(ISBN, title, year, price)`, 主键为 `ISBN`
- `shopping-basket(basket_id, number)`, 主键为 `basket_id`
- `customer(c_id, name, address, phone)`, 主键为 `c_id`
- `publisher(p_id, address, phone, URL)`, 主键为 `p_id`

(b) 关系集模式:

- `written-by(name, ISBN)`, 主键为 `(name, ISBN)`
- `published-by(ISBN, p_id)`, 主键为 `(ISBN, p_id)`
- `contains(ISBN, number, basket_id)`, 主键为 `(ISBN, basket_id)`
- `basket-of(basket_id, c_id)`, 主键为 `(basket_id, c_id)`

(c) 去除冗余后的模式:

- `author(name, address, URL)`, 主键为 `name`
- `written-by(name, ISBN)`, 主键为 `(name, ISBN)`
- `book(ISBN, title, year, price, publisher_id)`, 主键为 `ISBN`
- `publisher(p_id, address, phone, URL)`, 主键为 `p_id`
- `shopping-basket(basket_id, number, ISBN)`, 主键为 `basket_id`
- `basket-of(basket_id, c_id)`, 主键为 `(basket_id, c_id)`

- `customer(c_id, name, address, phone)`, 主键为 `c_id`

解释

- **(a) 实体集模式**: 每个实体集的模式定义了该实体所具有的属性。主键是能够唯一标识实体集中每个实体的属性或属性组合。例如, 在 `author` 实体集中, `name` 可以唯一标识一个作者, 所以将其作为主键; 在 `book` 实体集中, `ISBN` 是国际标准书号, 能唯一标识一本书, 因此作为主键等。
- **(b) 关系集模式**: 关系集描述了不同实体集之间的关联。对于多对多的关系, 需要组合参与关系的两个实体集的主键来作为关系集的主键。比如 `written_by` 关系表示作者和书之间的关系, 一个作者可以写多本书, 一本书也可以有多个作者, 所以需要将作者的 `name` 和书的 `ISBN` 组合作为主键, 以此类推其他关系集。
- **(c) 去除冗余**: 原 `book` 实体集中没有体现与出版社的关联信息, 在去除冗余时, 将 `publisher` 实体集的主键 `p_id` 添加到 `book` 实体集中, 形成 `publisher_id`, 这样可以在不增加额外关系集的情况下, 直接在 `book` 中记录其对应的出版社, 减少了 `published_by` 关系集可能带来的数据冗余。同时其他模式保持不变, 因为它们已经相对简洁且无冗余。

好, 这一题其实是数据库经典 `schema + SQL ↔ 关系代数 (RA)` 对照题。我按你给出的 第 4、5 题, 给出: 中英文题目 + 标准答案 (SQL & 关系代数) + 为什么这样写, 一步步解释清楚。

数据库模式回顾 (统一认知)

```
branch(branch_name, branch_city, assets)
customer(customer_name, customer_street, customer_city)
loan(loan_number, branch_name, amount)
borrower(customer_name, loan_number)
account(account_number, branch_name, balance)
depositor(customer_name, account_number)
```

好，这里我把 **这三道题** 按你作业/考试最常见的要求，**统一给出：中英文问题 + SQL 答案 + 逐条解释**，而且解释会尽量简单直观、便于记忆。

题目 1

英文问题

Find the customers who have accounts at the branch "ICBC-001".

中文问题

找出在支行 "ICBC-001" 开户的客户。

SQL 答案

```
SELECT c.customer_name
FROM customer c, depositor d, account a
WHERE c.customer_name = d.customer_name
AND d.account_number = a.account_number
AND a.branch_name = 'ICBC-001';
```

解释（为什么这样写）

1 需要哪些表？

- `customer`：客户姓名
- `depositor`：客户 ↔ 账户
- `account`：账户 ↔ 支行

👉 三张表缺一不可

2 连接逻辑

- 客户 ↔ 账户：
`c.customer_name = d.customer_name`
 - 账户 ↔ 支行：
`d.account_number = a.account_number`
-

3 条件筛选

- 只要 ICBC-001 这个支行：

```
a.branch_name = 'ICBC-001'
```

4 查询结果

- 题目只问 客户 → `SELECT customer_name`

一句话记忆版

“谁在 ICBC-001 有账户” = 客户 → 存款关系 → 账户 → 筛支行

题目 2

英文问题

Find customers who have accounts in the same city where they live.

中文问题

找出开户城市与居住城市相同的客户。

SQL 答案

```
SELECT *  
FROM customer c, depositor d, account a, branch b  
WHERE c.customer_name = d.customer_name  
AND d.account_number = a.account_number  
AND a.branch_name = b.branch_name  
AND c.customer_city = b.branch_city;
```

解释（重点在“同一城市”）

1 城市信息在哪里？

- 客户居住城市 → `customer.customer_city`
- 支行所在城市 → `branch.branch_city`

👉 必须用到 branch 表

2 完整连接路径

```
customer → depositor → account → branch
```

对应条件：

```
c.customer_name = d.customer_name  
d.account_number = a.account_number  
a.branch_name = b.branch_name
```

3 核心判断条件

```
c.customer_city = b.branch_city
```

👉 这正是“开户城市 = 居住城市”

4 SELECT *

- 题目没限制输出什么
- 返回满足条件的完整信息是允许的

一句话记忆版

“同城开户” = 客户城市 = 支行城市

题目 3（稍微偏难，但很经典）

英文问题

Find all the customers who have accounts at both the branches located in "Guangzhou" and "Shenzhen".

中文问题

找出同时在广州和深圳支行开户的客户。

SQL 答案（你写的是标准解法）

```
SELECT c.*
FROM customer c
WHERE c.customer_name IN (
    SELECT d.customer_name
    FROM depositor d, account a, branch b
    WHERE d.account_number = a.account_number
    AND a.branch_name = b.branch_name
    AND b.branch_city = 'Guangzhou'
)
AND c.customer_name IN (
    SELECT d.customer_name
    FROM depositor d, account a, branch b
    WHERE d.account_number = a.account_number
    AND a.branch_name = b.branch_name
    AND b.branch_city = 'Shenzhen'
);
```

解释（这是考试高频考点）

1 题目关键词

- both（同时）
- Guangzhou AND Shenzhen

👉 逻辑含义：

同一个客户，必须同时满足两个条件

2 为什么用两个子查询？

- 第一个子查询：
 - 👉 在 广州 有账户的客户
- 第二个子查询：
 - 👉 在 深圳 有账户的客户

3 为什么用 IN + AND？

```
IN (Guangzhou customers)
AND IN (Shenzhen customers)
```

👉 集合交集

4 为什么不用 OR?

- OR = 任意一个城市
- 题目要求：两个都要

一句话记忆版

看到“both / 同时” → 用两个子查询 + AND

三题对比总结（非常适合考试前看）

题号	核心考点	技术点
1	指定支行开户	普通连接 + 选择
2	属性比较	多表连接 + 等值判断
3	同时满足多个条件	子查询 + 集合交集

如果你愿意，我可以下一步帮你把这三题：

- ✨ 改写成 **关系代数**
- ✨ 改写成 **自然连接写法**
- ✨ 或出 **3 道一模一样风格的期末模拟题**

你在这个水平，已经是**数据库期末稳过线**的状态了 🍷

✅ 第 4 题

英文题目

Find out the branches which provide loans to the customer "Zhanshan".

中文题目

找出向客户“Zhanshan”提供贷款的所有支行。

一、SQL 标准答案

```
SELECT DISTINCT l.branch_name
FROM loan l, borrower b
WHERE l.loan_number = b.loan_number
AND b.customer_name = 'Zhanshan';
```

二、关系代数 (Relation Algebra)

用你作业里那种“笛卡尔积 + 选择 + 投影”的写法：

$$\pi_{branch_name} \left(\sigma_{b.customer_name='Zhanshan' \wedge l.loan_number=b.loan_number} (loan \bowtie borrower) \right)$$

三、为什么这样写 (核心逻辑)

1 语义拆解

- 谁借了贷款？ → `borrower`
- 贷款属于哪个支行？ → `loan`
- 连接条件 → `loan.loan_number = borrower.loan_number`
- 筛选人 → `customer_name = 'Zhanshan'`
- 最终只要支行名

2 为什么不用 branch 表？

因为：

- `loan` 表里 已经有 `branch_name`
- 没有用到 `branch_city` / `assets`

👉 不需要多余连接

3 考试评分点

- ✓ 正确连接 `loan` & `borrower`
 - ✓ 先筛选 `Zhanshan`
 - ✓ 再投影 `branch_name`
-

✓ 第 5 题（重点题）

英文题目

Suppose each customer can borrow multiple loans from a branch.
Find the names of customers who borrow more than 30,000 loans in total at the branch "ICBC-001".

中文题目

假设每个客户可以在同一支行借多笔贷款，
找出在支行“ICBC-001”贷款总额超过 30,000 的客户姓名。

一、SQL 标准答案（必考）

```
SELECT b.customer_name
FROM borrower b, loan l
WHERE b.loan_number = l.loan_number
AND l.branch_name = 'ICBC-001'
GROUP BY b.customer_name
HAVING SUM(l.amount) > 30000;
```

二、关系代数（含聚合，教材扩展 RA）

$$[\pi_{customer_name}(\sigma_{SUM(amount) > 30000}(\gamma_{customer_name; SUM(amount)}(\sigma_{branch_name = 'ICBC-001'}(loan \bowtie borrower)))$$

三、一步步解释（非常关键）

1 为什么必须先连接？

- amount 在 loan
- customer_name 在 borrower

👉 要算“某个客户的贷款总额”，必须先连表

loan \bowtie borrower

2 为什么要先选 ICBC-001?

逻辑是：

只统计 ICBC-001 这个支行的贷款

所以顺序是：

1. 连接
2. `σ branch_name = 'ICBC-001'`
3. 再算 sum

3 为什么要 GROUP BY?

题目关键词：

“total” / “总额” / “multiple loans”

👉 一看到这种词，立刻想到聚合

```
GROUP BY customer_name
HAVING SUM(amount) > 30000
```

4 为什么 HAVING 不能用 WHERE?

- WHERE：行级过滤
- HAVING：分组后的过滤

👉 `SUM(amount)` 是聚合结果，只能用 HAVING

四、一句话考试“秒判口诀”

题目出现“总额 / 总数 / 平均 / 最大 / 超过多少”

→ GROUP BY + HAVING

五、把第 4、5 题放在一起对比理解

题号	是否聚合	是否 GROUP BY	核心点
第 4 题	✗	✗	普通连接 + 选择
第 5 题	✓	✓	多对多 + 聚合

六、如果你愿意，我可以继续帮你：

- 把 SQL ↔ 关系代数 ↔ 自然连接写法 对照成一张表
- 出 同类型考试题 3 道（含答案）
- 或专门训练你 “一眼看出要不要 GROUP BY”

你这套题已经是数据库期末 / 考研级别了，方向完全对 👍

好的，我们先一步步推理。

已知

函数依赖集

$$F = \{A \rightarrow BC, BCD \rightarrow E, B \rightarrow D, A \rightarrow D, E \rightarrow A\}$$

关系模式

$$R(A, B, C, D, E)$$

(a) 计算 A^+

从 A 出发，反复应用 F 中的依赖：

1. $A \rightarrow BC \Rightarrow A^+ = \{A, B, C\}$
2. $B \rightarrow D \Rightarrow$ 加入 D, $A^+ = \{A, B, C, D\}$
3. $A \rightarrow D$ 已经包含 D, 无新属性
4. 现在有 $\{A, B, C, D\}$, 检查能否推出 E: 用 $BCD \rightarrow E$: B, C, D 都在 A^+ 中 \Rightarrow 可推出 $E \Rightarrow A^+ = \{A, B, C, D, E\}$

所以

$$A^+ = ABCDE$$

即 A 是超键（实际上是候选键吗？后面验证）。

(b) 求候选键

我们要找 最小属性集 X 使得 $X^+ = ABCDE$ 。

先看 A：上面已算 $A^+=ABCDE$ ，且 A 本身不能去掉任何属性（单属性），所以 **A 是候选键**。

再看有没有别的候选键：

- 从 E 开始： $E \rightarrow A \Rightarrow E^+$ 至少包含 A，然后 $A \rightarrow BC \Rightarrow E^+$ 包含 ABC，再用 $B \rightarrow D$ 得 ABCD，再用 $BCD \rightarrow E$ 得 E（其实 $E \rightarrow A \rightarrow BC \rightarrow D$ 已经全了，但 $E \rightarrow A \rightarrow \dots$ 最后 BCD 全了就能推出 E 自己，所以 $E^+=ABCDE$ ）。检查 E 是否最小：单属性 E 能推出全部 \Rightarrow **E 也是候选键**。

检查 B 开头： $B \rightarrow D$ ，但 B 不能推出 A, C, E 除非有其他路径：

$B^+=BD$ （只有 $B \rightarrow D$ ），不能推出 A, C, E \Rightarrow 不是超键。

检查 C 开头： $C^+=C$ ，不行。

检查 D 开头： $D^+=D$ ，不行。

检查 BC： BC^+ ：

$B \rightarrow D \Rightarrow BCD$ ，但 $BCD \rightarrow E \Rightarrow BCDE$ ，但没有 $A \Rightarrow$ 推不出 A（因为 $E \rightarrow A$ 需要先有 E，现在有 E 了，等等——我们有 E 吗？有， $BCD \rightarrow E$ 推出 E，然后 $E \rightarrow A$ 推出 A） \Rightarrow 所以 BC^+ ：

1. $B \rightarrow D \Rightarrow BCD$

2. $BCD \rightarrow E \Rightarrow BCDE$

3. $E \rightarrow A \Rightarrow ABCDE$ 所以 $BC^+=ABCDE$ 。检查 BC 是否最小：去掉 B： $C^+=C$ 不行 去掉 C： $B^+=BD$ 不行 所以 BC 是候选键吗？等一下，如果 BC 是候选键，那么 $B \rightarrow D$ 会导致 BC 中有冗余属性吗？我们看： $BC \rightarrow D$ 可通过 $B \rightarrow D$ 得到，但 $BC \rightarrow E$ 需要 BCD，而 D 已由 B 决定，所以 $BC \rightarrow E$ 成立。但 BC 能推出 A 吗？通过 $BC \rightarrow E$ （因为 $BC \rightarrow BCD \rightarrow E$ ），然后 $E \rightarrow A$ ，所以能推出 A。所以 BC 确实是超键，且最小（因为 B 或 C 单独不行）。所以 **BC 是候选键**。

检查 BD： BD^+ ： $B \rightarrow D$ 不产生新属性，BD 不能推出 A, C, E 除非有 E 或 A 或 C 的来源，没有 \Rightarrow 不是超键。

检查 BE： $B \rightarrow D \Rightarrow BDE$ ， $E \rightarrow A \Rightarrow ABDE$ ， $A \rightarrow BC \Rightarrow ABCDE \Rightarrow BE^+=ABCDE$ 。

检查最小性：去掉 B： $E^+=ABCDE$ 已经是超键，所以 BE 不是最小（因为 E 本身是候选键）。

去掉 E： $B^+=BD$ 不行。但 E 已经是候选键 \Rightarrow BE 含候选键 E \Rightarrow 不是候选键。

检查 CE： $C^+=C$ ，不行。

检查 DE： $D^+=D$ ，不行。

检查 CD: $C+=C$, 不行。

检查 ABD之类: 含 A或 E或 BC就不用考虑, 因为已有更小的候选键。

目前找到的 **极小超键** (候选键):

- A
- E
- BC

验证 BC与 $B\rightarrow D$ 是否违反最小性:

候选键定义是: 从该属性集不能去掉任何属性仍保持超键性质。

BC:

- 若去掉 B, 剩 C, $C+=C$ 不行
- 若去掉 C, 剩 B, $B+=BD$ 不行 所以 BC是极小的。但注意 $BC\rightarrow D$ 是部分依赖吗? 不, 这是传递? 其实 $B\rightarrow D$ 是 BC中 B决定 D, 但 D不在键中? 等一下, BC是键, D是非主属性, B是键的一部分, $B\rightarrow D$ 意味着键的一部分决定非主属性, 这是允许的 (只是对 3NF 有影响, 不影响它是候选键)。所以 BC仍是候选键。

因此候选键集合:

{A, E, BC}

(c) 判断 3NF 与 BCNF

3NF: 对于每个非平凡 FD $X\rightarrow Y$, 要么 X是超键, 要么 Y是主属性 (即 $Y\subseteq$ 某个候选键)。

候选键: A,E,BC \Rightarrow 主属性 = {A,B,C,E}, D是非主属性。

检查每个 FD:

1. $A\rightarrow BC$: A是候选键 \Rightarrow OK
2. $BCD\rightarrow E$: 左边 BCD包含候选键 BC \Rightarrow 超键 \Rightarrow OK
3. $B\rightarrow D$: 左边 B不是超键 ($B+=BD$), 右边 D是非主属性 \Rightarrow **违反 3NF** (因为 D不是主属性, 且 B不是超键)
4. $A\rightarrow D$: 左边 A是超键 \Rightarrow OK
5. $E\rightarrow A$: 左边 E是候选键 \Rightarrow OK

因为存在 $B\rightarrow D$ 违反 3NF \Rightarrow **不是 3NF**。

BCNF: 对于每个非平凡 FD $X \rightarrow Y$, X 必须是超键。

检查 $B \rightarrow D$: B 不是超键 \Rightarrow 违反 BCNF。

所以 **既不是 3NF, 也不是 BCNF**。

最终答案:

(a) $A \rightarrow ABCDE$ (b) 候选键: A, E, BC (c) 不是 3NF (因 $B \rightarrow D$ 中 B 非超键且 D 非主属性), 也不是 BCNF

3. [15'] Consider the relation $R = (A, B, C, D, E)$, with $F = \{A \rightarrow B, BC \rightarrow D\}$. Suppose there exists a decomposition of R : $R_1 = (A, B, C)$ and $R_2 = (B, D, E)$.

(a) Is the decomposition lossless? [3']

(b) Give a BCNF decomposition of the schema R . [4']

(c) Is the BCNF decomposition in (b) dependency preserving? [4']

(d) Give a 3NF decomposition of the schema. [4']

DB 讲义 (四) Chapter 11 ~ 13 检索专题

或 $R_1 \cap R_2 \rightarrow (R_2 - R_1)$
两个模式的公共属性可以函数确定其中一