华南理工大学《数据库系统》试卷2015（A）题目解析（含中英文对照）

Part I 填空题（每空1分，共20分）

1. 题目

- **英文**：The collection of information stored in the database at a particular moment is called an ____ of the database. The overall design of the database is called the database ____.
- **中文**：在特定时刻存储在数据库中的信息集合称为数据库的__。**数据库的整体设计称为数据库__。**
- **答案**：instance（实例）；schema（模式）
- **解释**：
  - 数据库实例（instance）是“动态数据集合”，反映数据库在某一时刻的实际数据状态，例如2025年12月18日10点数据库中存储的所有学生记录，就是该时刻的实例，数据会随增删改操作变化。

- 数据库模式（schema）是“静态结构设计”，定义了数据的组织结构、属性类型、关系约束等，例如学生表包含学号（SID）、姓名（LASTNAME）等属性的设计，是固定的框架，不随数据变化而改变（参考摘要1）。

2. 题目

- **英文**：A relation schema is in ____ normal form if for all $X \rightarrow A$ in F^+, at least one of the following holds: X is a super key for R; Each attribute in $A - X$ is contained in a candidate key for R.
- **中文**：若关系模式中所有在F^+（函数依赖闭包）中的函数依赖$X \rightarrow A$都满足以下条件之一，则该关系模式属于____范式：X是R的超键；$A - X$中的每个属性都包含在R的某个候选键中。
- **答案**：3NF（第三范式）
- **解释**：
  3NF是为解决关系模式中“非主属性对候选键的传递依赖”问题而定义的范式。其核心约束针对函数依赖$X \rightarrow A$：要么决定因素X是能唯一确定所有属性的超键，要么被决定属性A（除去X中已有的属性）是主属性（属于候选键）。满足3NF的关系模式可避免大部分数据冗余和更新异常（参考摘要2）。

3. 题目

- **英文**：Let R be a relation schema, R_1 and R_2 form a decomposition of R. Decomposition is a ____ if for all legal database instances r of R, the natural join of R_1 and R_2 (from r) equals r.
- **中文**：设R为关系模式，R_1和R_2构成R的一个分解。若对R的所有合法数据库实例r，由r导出的R_1和R_2的自然连接等于r，则该分解是____。
- **答案**：lossless（无损分解）
- **解释**：
  无损分解是关系模式分解的关键要求，核心是“分解后不丢失原关系信息”。例如原关系$R(ABC)$分解为$R_1(AB)$和$R_2(BC)$，若通过自然连接能完全恢复R的所有元组（无伪元组、无数据丢失），则为无损分解；反之，若连接结果出现原关系不存在的元组，则为有损分解。

4. 题目

- **英文**：In E-R model, an entity is represented by a set of __. **A** __ is an association among several entities.
- **中文**：在E-R模型中，实体由一组__**表示。**__是多个实体之间的关联。
- **答案**：Attribute（属性）；relationship（关系）

- **解释**：
  - 实体（Entity）是现实世界中的客观事物（如“学生”“教师”），属性（Attribute）是实体的特征，例如“学生”实体的属性包括学号（SID）、姓名（LASTNAME）等，通过属性描述实体的具体信息（参考摘要5）。
  - 关系（Relationship）用于表示实体间的关联，例如“学生-教师”之间的“被教授”关系、“班级-教室”之间的“使用”关系，反映实体间的业务逻辑。

5. 题目

- **英文**：Assume relation r has b_r blocks and relation s has b_s blocks, therefore, in the best case, only ____ block transfers would be required for $r \bowtie s$ (natural join).
- **中文**：假设关系r有b_r个块，关系s有b_s个块，因此在最佳情况下，执行r与s的自然连接仅需____次块传输。
- **答案**：$b_r + b_s$（r的块数加s的块数）
- **解释**：
  块传输次数是衡量数据库I/O代价的核心指标。在最佳情况下（如两个关系已按连接属性排序，且内存足够容纳其中一个关系的所有块），连接过程中仅需读取r的所有块（b_r次）和s的所有块（b_s次），无需重复读取，因此总块传输次数为$b_r + b_s$。

6. 题目

- **英文**：An ideal hash function is ____ and ____, the former requires that each bucket is assigned the same number of search-key values from the set of all possible values.
- **中文**：理想的哈希函数具有__**和**__特性，其中前者要求从所有可能的搜索键值集合中，每个桶（bucket）被分配到相同数量的搜索键值。
- **答案**：uniform（均匀性）；random（随机性）
- **解释**：
  - 均匀性（uniformity）是哈希函数的核心特性，确保搜索键值能均匀分布到各个桶中，避免某一桶中键值过多导致“哈希冲突”加剧，从而保证查询效率（参考摘要3）。
  - 随机性（randomness）指哈希函数的输出与输入之间无明显规律，即使输入相似（如“ABC”和“ABD”），输出的哈希值也差异较大，进一步减少冲突，同时提升安全性（如密码存储场景）。

7. 题目

- **英文**：To generate query-evaluation plans for an expression, we have to generate logically equivalent expressions using ____.

- **中文**：为一个（关系代数）表达式生成查询执行计划时，需使用____生成逻辑等价的表达式。
- **答案**：equivalence rules（等价规则）
- **解释**：
  等价规则是“不改变查询结果”的表达式变换规则，例如“选择操作提前”（$\sigma_\theta(R \bowtie S) = \sigma_\theta(R) \bowtie S$）、“连接交换律”（$R \bowtie S = S \bowtie R$）。通过这些规则，可将原始关系代数表达式转换为多个逻辑等价的形式，为后续选择最优执行计划提供基础（参考摘要4）。

8. 题目

- **英文**：Consider a B+-tree of order n, if there are K search-key values in the file, the path from the root to the leaf node is no longer than ____.
- **中文**：考虑一个阶为n的B+树，若文件中有K个搜索键值，则从根节点到叶节点的路径长度不超过____。
- **答案**：$\lceil \log_{\lceil n/2 \rceil} K \rceil$（以$\lceil n/2 \rceil$为底，$K$的对数向上取整）
- **解释**：
  B+树的阶n表示每个节点最多可存储$n-1$个搜索键值、有n个子节点。为保证树的“平衡性”，每个非根节点至少有$\lceil n/2 \rceil$个子节点。因此，树的高度（根到叶的路径长度）由最少子节点数和总键值数决定，公式$\lceil \log_{\lceil n/2 \rceil} K \rceil$可确保路径长度最小，从而保证查询效

率（如阶为10的B+树，1000个键值的路径长度仅需3）。

9. 题目

- **英文**：A transaction has the following properties: __, __, isolation and durability.
- **中文**：事务具有以下特性：__、__、隔离性（isolation）和持久性（durability）。
- **答案**：atomicity（原子性）； consistency（一致性）
- **解释**：
  事务的ACID特性是数据库并发控制和恢复的基础：
  - 原子性（atomicity）：事务是“不可分割的工作单元”，要么全部执行成功（提交），要么全部执行失败（回滚），例如转账事务中“扣款”和“收款”必须同时完成或同时不完成。
  - 一致性（consistency）：事务执行前后，数据库从一个一致状态过渡到另一个一致状态，例如转账前A有100元、B有50元，转账后A有80元、B有70元，总金额保持150元不变。

10. 题目

- **英文**：When the final statement of a transaction has been executed, the transaction enters the ____ state. After a transaction has been rolled back and the database has been restored to its previous state, the transaction enters the ____ state.
- **中文**：当事务的最后一条语句执行完毕后，事务进入__**状态。当事务回滚且数据库恢复到之前的状态后，事务进入**__状态。
- **答案**：partially committed（部分提交）；aborted（中止）
- **解释**：
  事务的生命周期包含多个状态：
  - 部分提交（partially committed）：事务所有语句执行完毕，但修改仍暂存在内存缓冲区，未写入磁盘，此时若发生故障，修改可能丢失。
  - 中止（aborted）：事务执行过程中出现错误或故障，通过回滚（undo）操作撤销所有已做修改，数据库恢复到事务开始前的状态，事务最终处于中止状态，无法继续执行。

11. 题目

- **英文**：A schedule S is ____ if a transaction T_j in S needs a data item previously written by a transaction T_i, then the commit operation of T_i appears before the commit operation of T_j.
- **中文**：若调度S中，事务T_j需要访问事务T_i之前写入的数据项，则T_i的提交操作在T_j的提交操作之前执行，那么调度S是____。
- **答案**：recoverable（可恢复的）
- **解释**：
  可恢复调度是为避免“级联回滚”而定义的调度特性。例如，若T_i写入数据后未提交，T_j读取该未提交数据（脏读），若T_i随后回滚，T_j也需回滚；而可恢复调度要求T_i先提交，T_j再提交，确保T_j读取的是已确认的数据，即使T_i回滚，也无需影响T_j。

12. 题目

- **英文**：____ attribute values or ____ attribute values are not atomic.
- **中文**：__**属性值或**__属性值不具有原子性。
- **答案**：Multivalued（多值）；composite（复合）
- **解释**：
  原子性指属性值不可再分：

- 多值属性（multivalued attribute）：一个实体的该属性对应多个值，例如“学生”的“联系方式”可能包含电话和邮箱，值不可再拆分为单一原子值。
- 复合属性（composite attribute）：属性可拆分为多个子属性，例如“地址”可拆分为“省份”“城市”“街道”，每个子属性都是独立的原子值。

13. 题目

- **英文**：A relation schema may have an attribute that corresponds to the primary key of another relation. The attribute is called a ____.
- **中文**：一个关系模式中可能包含一个与另一个关系模式的主键对应的属性，该属性称为____。
- **答案**：foreign key（外键）
- **解释**：
  外键是实现关系间关联的核心机制，用于维护数据库的“参照完整性”。例如，“学生表”（含属性SID、CID）中的CID对应“班级表”（主键为CID）的主键，CID就是“学生表”的外键，确保学生表中的CID值必须在班级表中存在，避免出现“不存在的班级”的学生记录。

Part II 简答题（共80分）

1. 数据库设计I（16分）

题目

- **英文**：Consider the following conditions:
  1. The STUDENT may be taught by one and only one teacher. The TEACHER may be instructor of one or more STUDENT.
  2. The TEACHER may be responsible for one and only one CLASS. The CLASS may be the responsibility of one and only one TEACHER.
  3. The CLASS may be made of one or more STUDENT. The STUDENT must be a member of one and only one CLASS.
  4. The CLASS must have one and only one ROOM. The ROOM may belong to one or more CLASS.
     Notes: Assume entity CLASS has attributes CID, CNAME; entity ROOM has attributes RID, LOCATION; entity STUDENT has attributes SID, LASTNAME, FIRSTNAME; entity TEACHER has attributes TID, TEACHERNAME, TITLE.
     a) [8 points] Construct an E-R diagram showing these relationships.
     b) [4 points] Construct appropriate relation

schemas for the above E-R diagram.
c) [4 points] Create an index std_index on the student relation with SID as the search key.

- **中文**：考虑以下条件：

1. 一名学生（STUDENT）仅由一名教师（TEACHER）教授，一名教师可教授多名学生。
2. 一名教师仅负责一个班级（CLASS），一个班级仅由一名教师负责。
3. 一个班级包含多名学生，一名学生必须属于一个班级。
4. 一个班级必须使用一个教室（ROOM），一个教室可被多个班级使用。
说明：实体CLASS属性为CID（班级ID）、CNAME（班级名称）；实体ROOM属性为RID（教室ID）、LOCATION（位置）；实体STUDENT属性为SID（学号）、LASTNAME（姓）、FIRSTNAME（名）；实体TEACHER属性为TID（教师ID）、TEACHERNAME（教师姓名）、TITLE（职称）。
a) [8分] 绘制表示上述关系的E-R图。
b) [4分] 为上述E-R图构造合适的关系模式。
c) [4分] 在学生关系上以SID为搜索键创建索引std_index。

答案与解释

a) E-R图构造（文字描述版）

- **实体（矩形）**：
  - STUDENT（属性：SID（主键）、LASTNAME、FIRSTNAME）
  - TEACHER（属性：TID（主键）、TEACHERNAME、TITLE）
  - CLASS（属性：CID（主键）、CNAME）
  - ROOM（属性：RID（主键）、LOCATION）
- **关系（菱形）及基数约束**：

1. **Teaches（TEACHER 与 STUDENT）**
   - **一个 TEACHER 教多少 STUDENT?** → 多（N）
   - **一个 STUDENT 被多少 TEACHER 教?** → 1
     ✅ 标注：**TEACHER (1) —— STUDENT (N)**

---

1. **Manages（TEACHER 与 CLASS）**
   - **一个 TEACHER 管理多少 CLASS?** → 1
   - **一个 CLASS 由多少 TEACHER 管理?** → 1
     ✅ 标注：**TEACHER (1) —— CLASS (1)**

---

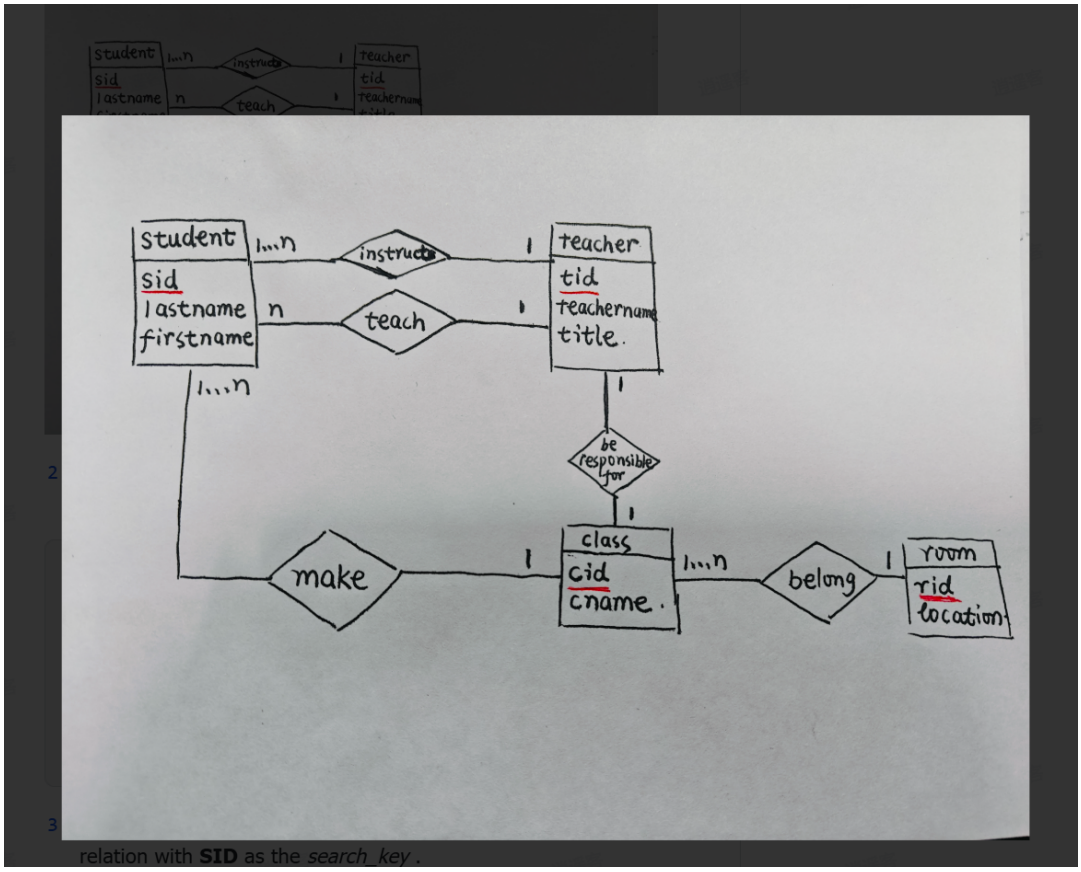
1. **Belongs_to（STUDENT 与 CLASS）**
   - **一个 STUDENT 属于多少 CLASS?** → 1

- **一个 CLASS 有多少 STUDENT?** → 多（N）
  ✅ 标注：**CLASS (1) —— STUDENT (N)**

> ⚠️ 注意：这里**不能写成“STUDENT 1 — CLASS N”**，之前你的写法标反了

---

1. **Uses（CLASS 与 ROOM）**

- **一个 CLASS 使用多少 ROOM?** → 1
- **一个 ROOM 被多少 CLASS 使用?** → 多（N）
  ✅ 标注：**CLASS (1) —— ROOM (N)**
- 

b) 关系模式构造

关系模式设计需将E-R图中的实体和关系转换为关系，通过外键维护关联：

1. **STUDENT(SID, LASTNAME, FIRSTNAME, TID, CID)**

- 主键：SID；外键：TID（参照TEACHER.TID）、CID（参照CLASS.CID）
- 说明：通过TID关联教师（Teaches关系），通过CID关联班级（Belongs_to关系）。

2. **TEACHER(TID, TEACHERNAME, TITLE, CID)**
   - 主键：TID；外键：CID（参照CLASS.CID）
   - 说明：通过CID关联负责的班级（Manages关系，1对1，故CID可作为TEACHER的外键）。
3. **CLASS(CID, CNAME, RID)**
   - 主键：CID；外键：RID（参照ROOM.RID）
   - 说明：通过RID关联使用的教室（Uses关系）。
4. **ROOM(RID, LOCATION)**
   - 主键：RID（无外键，为独立实体）。

   也可以

```
student(sid, lastname, firstname, tid)
instrcution(sid, tid)
teacher(tid, teachername, title, cid)
class(cid, cname, rid)
room(rid, location)
make(sid, cid)
```

c) 创建索引的SQL语句

- **英文**:

```sql
CREATE INDEX std_index ON STUDENT(SID);
```

- **中文解释**：
  该SQL语句在STUDENT关系的SID属性上创建名为std_index的索引。索引的作用是加速基于SID的查询（如“查询SID=2025001的学生信息”），避免全表扫描。由于SID是主键，本质上数据库会自动为其创建唯一索引，但此处按题目要求显式创建普通索引（若需唯一约束，可添加UNIQUE关键字）。

2. 数据库设计中的关系集转换（6分）

题目

- **英文**：In database design, how to represent relationship set as relational schema?
- **中文**：在数据库设计中，如何将关系集（relationship set）表示为关系模式?

答案与解释

将E-R模型中的关系集转换为关系模式，需根据关系的“基数约束”（1对1、1对多、多对多）采用不同策略，核心是通过“外键”或“独立关系表”维护实体间关联，具体步骤如下：

1. **1对1（1:1）关系集**

- 方法：选择任一关联实体的关系模式，添加另一实体的主键作为外键。
- 示例：TEACHER与CLASS的1:1关系“Manages”，可在TEACHER表中添加CLASS的主键CID作为外键（如TEACHER(TID, TEACHERNAME, TITLE, CID)），无需创建独立关系表。
- 理由：1:1关系中，两个实体的主键可互相唯一标识对方，添加外键即可体现关联，避免冗余。

2. **1对多（1:N）关系集**
   - 方法：在“多”端实体的关系模式中，添加“1”端实体的主键作为外键。
   - 示例：TEACHER与STUDENT的1:N关系“Teaches”，在STUDENT表中添加TEACHER的主键TID作为外键（如STUDENT(SID, LASTNAME, TID)）。
   - 理由：“多”端实体的每个元组仅对应“1”端的一个元组，外键可直接关联，无需独立表。
3. **多对多（M:N）关系集**
   - 方法：创建**独立的关系表**，表中包含所有关联实体的主键（作为联合主键），若关系有属性，也需纳入表中。

- 示例：STUDENT与COURSE的M:N关系“Takes”（学生选多门课，一门课被多学生选），需创建Takes(SID, CNO, Grade)，其中SID（参照STUDENT.SID）、CNO（参照COURSE.CNO）为联合主键，Grade为关系的属性（成绩）。
- 理由：M:N关系中，无法通过单一实体的外键体现关联，独立表可存储多对多的映射关系及关系自身属性。

翻译结果

在数据库设计中，将关系集表示为关系模式需创建一种表结构，该结构能以符合关系型数据库设计原则的方式捕捉实体及其之间的关联。以下是将关系集表示为关系模式的分步指南：

1. **识别实体与属性**：
   - 确定参与该关系的所有实体；
   - 识别描述每个实体的属性（列）。
2. **确定关系类型**：
   - 明确关系的性质（一对一、一对多、多对多）；
   - 这将有助于确定关系模式的结构。
3. **创建主键**：
   - 为每个实体分配唯一标识符（主键），确保每条记录都能被唯一标识。

4. **定义外键**：
    - 对于实体间的关系，在关联表中使用外键来维护参照完整性；
    - 一个表中的外键指向另一个表的主键。
5. **数据库规范化**：
    - 应用规范化规则以消除数据冗余，确保数据完整性；
    - 其目标是减少数据异常，提升数据库的完整性。
6. **创建数据表**：
    - 基于实体及其间的关系，为每个实体创建对应的数据表；
    - 根据需要包含主键和外键。
7. **实现关系**：
    - 对于一对多关系，将外键放置在关系中“多”的一侧的表中；
    - 对于多对多关系，创建一个关联（连接）表，该表包含两个相关实体的主键。

翻译说明

1. **术语精准性**：
    - `relationship set` 译为“关系集”（数据库领域标准术语）；

- `relational schema` 译为“关系模式”（区别于“schema”在其他场景的“架构”译法）；
- `primary key/foreign key` 译为“主键/外键”、`referential integrity` 译为“参照完整性”、`normalization` 译为“规范化”、`associative (junction) table` 译为“关联（连接）表”，均遵循数据库行业通用译法。

2. **句式适配**：
   - 英文长句拆分为符合中文表达习惯的短句（如开篇“In database design...principles of relational database design”），避免直译导致的语句臃肿；
   - 保留步骤式结构，序号与原文一一对应，逻辑清晰；
   - 被动语态（如“is placed”）转为主动语态（“放置在”），符合中文主动表达偏好。
3. **语义完整性**：
   - “eliminate redundancy and ensure data integrity” 译为“消除数据冗余，确保数据完整性”，完整覆盖“冗余消除”和“完整性保障”两层含义；
   - “reduce data anomalies” 译为“减少数据异常”，而非字面直译“减少数据反常现象”，贴合数据库场景中“数据异常（插入/更新/删除异常）”的核心含义。

3. 关系模式范式分析（14分）

题目

- **英文**：Let $R = (A, B, C, D, E, F)$ be a relation with functional dependency $F = \{A \to CB, E \to FA\}$.
  a) [2 points] Compute the candidate keys for R.
  b) [6 points] Is R in 3NF? If it is, justify your answer. If not, produce a decomposition of R into 3NF.
  c) [6 points] Is R in BCNF? If it is, justify your answer. If not, produce a decomposition of R into BCNF.
- **中文**：设关系模式$R = (A, B, C, D, E, F)$，函数依赖集$F = \{A \to CB, E \to FA\}$。
  a) [2分] 计算R的候选键。
  b) [6分] R是否属于3NF？若是，说明理由；若否，将R分解为3NF。
  c) [6分] R是否属于BCNF？若是，说明理由；若否，将R分解为BCNF。

答案与解释

a) 候选键计算

候选键是“能唯一确定所有属性的最小属性集”，通过**属性闭包**计算：

1. **确定属性分类**：
   - 左部属性（L）：仅出现在函数依赖左部的属性，此处为D, E（A出现在左部和右部，B, C, F仅出现在右部）。
   - 必须包含L属性：候选键需包含D和E（否则无法确定D，因无依赖导出D）。
2. **计算$(ED)^+$（ED的属性闭包）**：
   - 初始：$(ED)^+ = \{E, D\}$
   - 由$E \to FA$，添加F, A：
     $(ED)^+ = \{E, D, F, A\}$
   - 由$A \to CB$，添加C, B：
     $(ED)^+ = \{E, D, F, A, C, B\}$（覆盖所有属性）。
3. **验证最小性**：
   - 移除E：$D^+ = \{D\}$（无法覆盖其他属性）。
   - 移除D：$E^+ = \{E, F, A, C, B\}$（无法覆盖D）。
     故ED是唯一候选键。

b) 3NF判定与分解

1. **3NF定义**：对所有$X \to A \in F^+$，满足：X是超键，或A是主属性（属于候选键）（参考摘要2）。
2. **判定R是否为3NF**：

- 主属性：E, D（属于候选键ED）；非主属性：A, B, C, F。
- 检查函数依赖：
  - $A \to CB$：A不是超键（$A^+ = \{A, C, B\}$无法覆盖所有属性），且C, B是非主属性→违反3NF。
  - $E \to FA$：E不是超键（$E^+ = \{E, F, A, C, B\}$无法覆盖D），且F, A是非主属性→违反3NF。
    故R不属于3NF。

3. **3NF分解（基于 canonical cover ）**：
   - 步骤1：求F的规范覆盖（canonical cover）F_c：$F_c = \{A \to CB, E \to FA\}$（无需简化）。
   - 步骤2：为每个依赖创建关系模式：
     - $R_1(A, B, C)$（对应$A \to CB$）。
     - $R_2(E, F, A)$（对应$E \to FA$）。
   - 步骤3：添加包含候选键的关系模式（确保无损分解）：
     - $R_3(E, D)$（包含候选键ED的核心属性）。
       最终分解：$R_1(A, B, C)$、$R_2(E, F, A)$、$R_3(E, D)$（均满足3NF）。

c) BCNF判定与分解

1. **BCNF定义**：对所有非平凡函数依赖$X \to A \in F^+$，X必须是超键（参考摘要6）。
2. **判定R是否为BCNF**：
   - 检查函数依赖：
     - $A \to CB$：A不是超键→违反BCNF。
     - $E \to FA$：E不是超键→违反BCNF。
       故R不属于BCNF。
3. **BCNF分解（递归分解法）**：
   - 步骤1：选择违反BCNF的依赖$A \to CB$，分解为：
     - $R_1(A, B, C)$（A是主键，满足BCNF）。
     - $R_2(A, D, E, F)$（包含A及剩余属性，需进一步检查）。
   - 步骤2：检查R_2的依赖$E \to FA$（E不是超键），分解为：
     - $R_3(E, F, A)$（E是主键，满足BCNF）。
     - $R_4(E, D)$（E是主键，满足BCNF）。
       最终分解：$R_1(A, B, C)$、$R_3(E, F, A)$、$R_4(E, D)$（均满足BCNF）。

4. 数据库查询与操作（28分）

题干涉及的关系表说明

首先明确题干中7个关系表的核心信息（便于理解查询逻辑）：

表名	核心属性及作用
BOOK	Bookid（图书ID，主键）、Title（书名）、Publishername（出版社名）→ 存储图书基本信息
BOOK_AUTHORS	Bookid（图书ID，外键）、Authorname（作者名）→ 存储图书与作者的多对多关系
PUBLISHER	Publishername（出版社名，主键）、Address（地址）、Phone（电话）→ 存储出版社信息

表名	核心属性及作用
BOOK_COPIES	Bookid（图书ID，外键）、Branchid（分馆ID，外键）、No_Of_Copies（副本数量）→ 存储分馆图书副本数
LIBRARY_BRANCH	Branchid（分馆ID，主键）、Branchname（分馆名）、Address（地址）→ 存储图书馆分馆信息
BOOK_LOANS	Bookid（图书ID，外键）、Branchid（分馆ID，外键）、Cardno（借阅卡号，外键）、DataOut（借出日期）、Duedata（到期日期）→ 存储图书借阅记录
BORROWER	Cardno（借阅卡号，主键）、Name（借阅人姓名）、Address（地址）、Phone（电话）→ 存储借阅人信息

一、中英文问题+答案+解释（按题目模块梳理）

模块a：SQL查询表达（Q1-Q4）

Q1

- **英文问题**：How many copies of the book titled The Lost Tribe are owned by the library branch whose name is “Sharpstown”?
- **中文问题**：名称为“Sharpstown”的图书馆分馆拥有多少本标题为《The Lost Tribe》的图书副本?
- **SQL答案**：

```
SELECT bc.No_Of_Copies
FROM BOOK b
JOIN BOOK_COPIES bc ON b.Bookid =
bc.Bookid
JOIN LIBRARY_BRANCH lb ON bc.Branchid =
lb.Branchid
WHERE b.Title = 'The Lost Tribe' AND
lb.Branchname = 'Sharpstown';
```

- **解释**：

1. 表关联逻辑：`BOOK`（图书基本信息）、`BOOK_COPIES`（图书副本数量）、`LIBRARY_BRANCH`（分馆信息）通过主键/外键关联（`Bookid`关联`BOOK`和`BOOK_COPIES`，`Branchid`

关联`BOOK_COPIES`和`LIBRARY_BRANCH`）；
2. 筛选条件：指定图书标题为“The Lost Tribe”、分馆名称为“Sharpstown”；
3. 查询目标：提取符合条件的副本数量`No_Of_Copies`。

Q2

- **英文问题**：For each library branch, retrieve the branch name and the total number of books loaned out from that branch.
- **中文问题**：对于每个图书馆分馆，检索该分馆的名称以及从该分馆借出的图书总数。
- **SQL答案**：

```
SELECT lb.Branchname, COUNT(bl.Bookid) AS
Total_Loaned_Books
FROM LIBRARY_BRANCH lb
LEFT JOIN BOOK_LOANS bl ON lb.Branchid =
bl.Branchid
GROUP BY lb.Branchname;
```

- **解释**：
1. 表关联逻辑：用`LEFT JOIN`关联`LIBRARY_BRANCH`和`BOOK_LOANS`（确保无借出记录的分馆也会显示，总数为0）；

2. 聚合与分组：按分馆名称`Branchname`分组，通过`COUNT(bl.Bookid)`统计每组（分馆）的借出图书数量；
3. 注意：原参考答案中错误关联`BOOK_COPIES`（副本数≠借出数），修正为关联`BOOK_LOANS`（借出记录）更符合题意。

Q3

- **英文问题**：Retrieve the name, address, and number of books checked out for all borrowers who have more than five books checked out.
- **中文问题**：检索所有借出图书数量超过5本的借阅者的姓名、地址，以及他们借出的图书数量。
- **SQL答案**：

```
SELECT bor.Name, bor.Address,
COUNT(bl.Bookid) AS Checked_Out_Count
FROM BORROWER bor
JOIN BOOK_LOANS bl ON bor.Cardno =
bl.Cardno
GROUP BY bor.Cardno, bor.Name,
bor.Address
HAVING COUNT(bl.Bookid) > 5;
```

- **解释**：

1. 表关联逻辑：`BORROWER`（借阅者信息）和`BOOK_LOANS`（借出记录）通过`Cardno`（借阅卡号）关联；
2. 分组与筛选：按借阅者唯一标识`Cardno`（及姓名、地址）分组，用`HAVING`过滤出借出数量>5的分组；
3. 注意：`GROUP BY`需包含非聚合字段（`Name`/`Address`），确保语法合规（不同数据库对`GROUP BY`宽松度不同，显式包含更通用）。

Q4

- **英文问题**：For each book authored (or co-authored) by “Stephen King”, retrieve the title and the number of copies owned by the library branch whose name is “Central”.
- **中文问题**：对于每本由“Stephen King”独著或合著的图书，检索其标题以及名称为“Central”的图书馆分馆拥有的该图书副本数量。
- **SQL答案**：

```sql
SELECT b.Title, bc.No_Of_Copies
FROM BOOK_AUTHORS ba
JOIN BOOK b ON ba.Bookid = b.Bookid
JOIN BOOK_COPIES bc ON b.Bookid =
bc.Bookid
JOIN LIBRARY_BRANCH lb ON bc.Branchid =
lb.Branchid
WHERE ba.Authorname = 'Stephen King' AND
lb.Branchname = 'Central';
```

- **解释**：
  1. 表关联逻辑：`BOOK_AUTHORS`（图书-作者关联）→`BOOK`（图书标题）→`BOOK_COPIES`（副本数）→`LIBRARY_BRANCH`（分馆名称）依次关联；
  2. 筛选条件：指定作者为“Stephen King”、分馆名称为“Central”；
  3. 查询目标：提取符合条件的图书标题和副本数量。

模块b：删除图书“T&G”的所有相关信息

- **英文问题**：Record the fact that the manager didn’t maintain information about the book named “T&G”, i.e. remove information about “T&G”.
- **中文问题**：记录管理员未维护名为“T&G”的图书信息这一事实（即删除所有与“T&G”相关的信息）。
- **SQL答案**：

```sql
-- 先删除关联表（外键约束需按依赖顺序删除）
DELETE FROM BOOK_AUTHORS WHERE Bookid IN
(SELECT Bookid FROM BOOK WHERE Title =
'T&G');
DELETE FROM BOOK_COPIES WHERE Bookid IN
(SELECT Bookid FROM BOOK WHERE Title =
'T&G');
DELETE FROM BOOK_LOANS WHERE Bookid IN
(SELECT Bookid FROM BOOK WHERE Title =
'T&G');
-- 最后删除主表BOOK中的记录
DELETE FROM BOOK WHERE Title = 'T&G';
```

- **解释**：

  1. 执行顺序：因`BOOK_AUTHORS`/`BOOK_COPIES`/`BOOK_LOANS`均依赖`BOOK`的`Bookid`（外键），需先删除关联表中相关记录，再删除主表`BOOK`的记录，避免外键约束报错；
  2. 子查询逻辑：通过`SELECT Bookid FROM BOOK WHERE Title = 'T&G'`定位“T&G”的图书ID，再批量删除关联表中该ID的记录；
  3. 最终删除`BOOK`表中标题为“T&G”的行，彻底清除该图书的所有信息。

模块A：BOOK_AUTHORS表的SQL DDL声明

- **英文问题**：Write appropriate SQL DDL statements for declaring the BOOK_AUTHORS relation.
- **中文问题**：编写用于声明BOOK_AUTHORS关系的合适SQL DDL语句。
- **SQL答案**：

```
CREATE TABLE BOOK_AUTHORS (
    Bookid CHAR(20),
    Authorname CHAR(200),
    -- 可选：添加外键约束（关联BOOK表）

    FOREIGN KEY (Bookid) REFERENCES
BOOK(Bookid)
);
```

- **解释**：

1. 字段定义：`Bookid`（图书ID，字符型，长度20）、`Authorname`（作者姓名，字符型，长度200），匹配题目给出的基础结构；
2. 约束补充：建议添加外键约束`FOREIGN KEY (Bookid) REFERENCES BOOK(Bookid)`，确保`Bookid`必须存在于`BOOK`表中，符合关系数据库的参照完整性。

模块B：关系代数表达（Q1-Q2）

Q1

- **英文问题**：Retrieve the names of all borrowers who do not have any books checked out.
- **中文问题**：检索所有没有借出任何图书的借阅者姓名。
- **关系代数答案**：

```
Temp ← ΠCardno(BORROWER) -
ΠCardno(BOOK_LOANS)
Res ← ΠName(Temp × BORROWER WHERE
Temp.Cardno = BORROWER.Cardno)
-- 简化写法：Res ← ΠName(BORROWER -
(BORROWER × BOOK_LOANS WHERE
BORROWER.Cardno = BOOK_LOANS.Cardno))
```

- **解释**：

1. 第一步（Temp）：通过“差集”运算，从所有借阅者卡号（`ΠCardno(BORROWER)`）中减去有借出记录的卡号（`ΠCardno(BOOK_LOANS)`），得到无借出记录的卡号集合；
2. 第二步（Res）：将无借出记录的卡号集合与`BORROWER`表做自然连接（按`Cardno`匹配），再投影（`Π`）出姓名`Name`，即得到目标结果。

Q2

- **英文问题**：For each book that is loaned out from the “Sharpstown” branch and whose DueDate is today, retrieve the book title, the borrower’s name, and the borrower’s address.
- **中文问题**：对于每本从“Sharpstown”分馆借出且到期日为今天的图书，检索图书标题、借阅者姓名和借阅者地址。
- **关系代数答案**：

```
ΠTitle,Name,Address (
    σBranchname="Sharpstown" ∧
DueDate=CURRENT_DATE (
        LIBRARY_BRANCH × BOOK_LOANS ×
BORROWER × BOOK
        WHERE
LIBRARY_BRANCH.Branchid=BOOK_LOANS.Branch
id
          AND
BOOK_LOANS.Cardno=BORROWER.Cardno
          AND
BOOK_LOANS.Bookid=BOOK.Bookid
    )
)
```

- **解释**：

1. 选择（σ）：先筛选出分馆名称为“Sharpstown”且到期日为今天（`CURRENT_DATE`）的记录；

2. 笛卡尔积+连接条件：将`LIBRARY_BRANCH`（分馆）、`BOOK_LOANS`（借出记录）、`BORROWER`（借阅者）、`BOOK`（图书）做笛卡尔积，并通过等值条件（`Branchid`/`Cardno`/`Bookid`）实现自然连接；
3. 投影（∏）：从筛选后的结果中提取`Title`（图书标题）、`Name`（借阅者姓名）、`Address`（借阅者地址）。

模块C：重复SQL查询（与模块a一致，修正原参考答案错误）

> 注：原参考答案中Q2错误关联`BOOK_COPIES`（副本数≠借出数）、Q3缺少`Name/Address`在`GROUP BY`中，已在模块a中修正并解释，此处不再重复。

二、关键补充说明

1. **表关联原则**：关系数据库中多表查询需通过主键-外键关联，避免笛卡尔积冗余；
2. **聚合查询**：`GROUP BY`需包含所有非聚合字段，`HAVING`用于过滤聚合结果（`WHERE`过滤行，`HAVING`过滤分组）；
3. **删除操作**：外键约束下需按“从表→主表”顺序删除，避免违反参照完整性；

4. **关系代数核心**：投影（∏）提取列、选择（σ）过滤行、笛卡尔积（×）+等值条件实现连接、差集（-）实现“不存在”逻辑。

5. Query Processing, Optimization and Transaction（查询处理、优化与事务）

a) 选择操作的实现过程及最佳情况开销分析

- **英文问题**：Please describe the implementation process of selection operation $\sigma_{A=c}(r)$, where r is a relation, A is an attribute and is not a candidate key, r has a primary index on A. If there are n matching records, the B+ tree index is of height h, and each disk block contains at most d records, please analyze the overhead in the best case.
- **中文问题**：请描述选择操作 $\sigma_{A=c}(r)$ 的实现过程（其中 r 是一个关系，A 是一个属性且不是候选键，r 在 A 上有主索引）。若有 n 条匹配记录，B+树索引的高度为 h，每个磁盘块最多包含 d 条记录，请分析最佳情况下的开销。

- **答案（实现过程）**：
  1. **索引查找**：通过B+树索引定位属性 A 等于 c 的记录：从B+树的根节点开始，逐层遍历到叶子节点（共访问 h 个索引块）；
  2. **记录读取**：从B+树叶子节点获取匹配记录的物理地址，读取对应的记录块。
- **答案（最佳情况开销）**：
  最佳情况是 n 条匹配记录存储在**同一个数据块**中（即 $n \leq d$），此时开销为：
  - 索引块访问次数：h;
  - 数据块访问次数：1;
  - 总磁盘块访问次数：$\boxed{h+1}$。

b) 索引嵌套循环连接（Indexed nested-loop join）的过程

- **英文问题**：Describe the process of Indexed nested-loop join.
- **中文问题**：描述索引嵌套循环连接的过程。
- **答案**：
  索引嵌套循环连接是针对“一个表（外表）小、另一个表（外表）在连接属性上有索引”的场景，过程如下：
  1. **遍历外表**：逐行读取外表（通常是较小的表）的每条记录 t;

2. **索引查找内表**：利用内表在连接属性上的索引，查找内表中与 t 的连接属性值匹配的所有记录；
3. **连接结果**：将外表记录 t 与内表中匹配的所有记录进行连接，生成结果集。

- **解释**：
  相比朴素嵌套循环连接，索引嵌套循环连接避免了对内表的全表扫描，通过索引快速定位匹配记录，降低了内表的访问开销。

c) 两段锁协议（Two-phase locking protocol）的描述，以及其保证冲突可串行化、不保证无死锁的证明

- **英文问题**：Please describe the two-phase locking protocol and prove that it ensures conflict-serializable schedules and does not ensure freedom from deadlocks.
- **中文问题**：请描述两段锁协议，并证明它保证冲突可串行化调度，但不保证无死锁。
- **答案（两段锁协议描述）**：
  两段锁协议要求事务的加锁、解锁操作分为两个阶段：
  1. **生长阶段（Growing phase）**：事务只能加锁，不能解锁；
  2. **收缩阶段（Shrinking phase）**：事务只能解锁，不能加锁。

- **答案（证明：保证冲突可串行化）**：
  对遵循两段锁协议的调度，按**事务的解锁顺序**排序，可得到冲突等价的串行调度：
  假设事务 T_1 先于 T_2 解锁，则 T_1 的加锁操作全部发生在 T_2 的加锁操作之前（因为两段锁的生长阶段不能解锁）。若 T_1 与 T_2 有冲突操作（如 T_1 写 X、T_2 读 X），则 T_1 的操作必然先于 T_2，符合冲突可串行化的“优先图无环”条件。

- **答案（证明：不保证无死锁）**：
  两段锁协议可能导致死锁：例如，事务 T_1 加锁 X，事务 T_2 加锁 Y；随后 T_1 尝试加锁 Y（处于生长阶段），T_2 尝试加锁 X（处于生长阶段），此时 T_1 与 T_2 互相等待对方的锁，形成死锁。

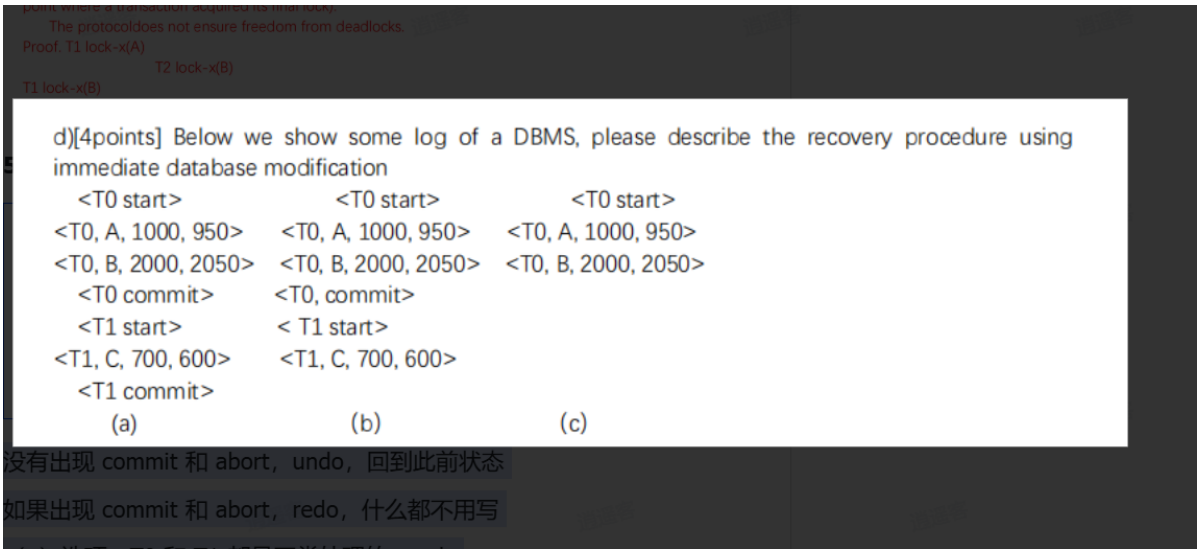
问题、答案与解释

英文问题

d) Below we show some logs of a DBMS, please describe the recovery procedure using immediate database modification for each case (a), (b), (c).

中文问题

d) 以下展示了DBMS的部分日志，请使用“即时数据库修改”策略描述每个情况（a）、（b）、（c）的恢复过程。



日志格式说明

日志条目格式为 `<事务名，数据项，旧值，新值>`，代表事务对数据项的修改（旧值→新值）；`<事务名 start>` 表示事务开始；`<事务名 commit>` 表示事务提交。

情况(a)的恢复过程

- **日志内容**：
  `<T0 start>` → `<T0, A, 1000, 950>` → `<T0, B, 2000, 2050>` → `<T0 commit>` → `<T1 start>` → `<T1, C, 700, 600>` → `<T1 commit>`

- **英文答案**:
  Both T0 and T1 have committed (their `commit` entries exist in the log). We perform **redo** for T0 and T1:
  - Set A to 950 (redo T0's modification to A)
  - Set B to 2050 (redo T0's modification to B)
  - Set C to 600 (redo T1's modification to C)
- **中文答案**:
  T0和T1均已提交（日志中存在它们的`commit`条目），因此对T0和T1执行**重做（redo）**：
  - 将A设为950（重做T0对A的修改）
  - 将B设为2050（重做T0对B的修改）
  - 将C设为600（重做T1对C的修改）
- **解释**:
  即时数据库修改中，已提交事务的修改需保留，因此通过redo操作确保这些修改生效。

情况(b)的恢复过程

- **日志内容**:
  `<T0 start>` → `<T0, A, 1000, 950>` → `<T0, B, 2000, 2050>` → `<T0 commit>` → `<T1 start>` → `<T1, C, 700, 600>`
- **英文答案**:

    - T0 has committed: perform **redo** for T0 → Set A to 950, B to 2050.
    - T1 has started but not committed (no `commit` entry): perform **undo** for T1 → Restore C to 700 (its old value).
- **中文答案**:
    - T0已提交：对T0执行**重做（redo）** → 将A设为950，B设为2050。
    - T1已开始但未提交（日志中无`commit`条目）：对T1执行**撤销（undo）** → 将C恢复为旧值700。
- **解释**:
  已提交的T0修改需保留（redo）；未提交的T1修改可能未持久化或需回滚，因此通过undo恢复数据项的旧值。

情况(c)的恢复过程

- **日志内容**:
  `<T0 start>` → `<T0, A, 1000, 950>` → `<T0, B, 2000, 2050>`
- **英文答案**:
  T0 has started but not committed (no `commit` entry). Perform **undo** for T0:
    - Restore A to 1000 (its old value)
    - Restore B to 2000 (its old value)

- **中文答案**：
  T0已开始但未提交（日志中无`commit`条目），对T0执行**撤销（undo）**：
  - 将A恢复为旧值1000
  - 将B恢复为旧值2000
- **解释**：
  未提交的事务T0的修改需全部回滚，因此通过undo操作将数据项恢复到修改前的状态。

要不要我帮你整理一份**即时数据库修改恢复策略的核心规则表**？