数据库系统英文试题（2013版）中英文对照+答案+详细解释

一、填空题（Fill-in-the-Blank Questions）

1.

- **英文题目**：A __ is a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.
- **中文题目**：__是一组概念性工具，用于描述数据、数据关系、数据语义和一致性约束。
- **答案**：Data Model（数据模型）
- **解释**：数据模型是数据库系统的核心基础，通过标准化的概念和工具，定义数据的组织形式、数据间的关联规则、数据的含义及数据必须满足的完整性要求，常见的有层次模型、网状模型、关系模型等。

2.

- **英文题目**：The three levels of data abstraction in database system include: __ level, __ level and view level.
- **中文题目**：数据库系统中的三级数据抽象包括：__层、__层和视图层。
- **答案**：Physical（物理）; Logical（逻辑）
- **解释**：三级数据抽象是数据库系统的重要特性，目的是隔离数据的不同层面，方便用户和系统管理：
  - 物理层（Physical Level）：描述数据的物理存储结构（如磁盘上的文件组织、索引存储方式）；
  - 逻辑层（Logical Level）：描述数据库的整体逻辑结构（如关系模式、属性、函数依赖），是数据库管理员关注的层面；
  - 视图层（View Level）：面向用户，仅展示用户所需的部分数据，屏蔽无关细节。

3.

- **英文题目**：If every entity in entity set E participates in at least one relationship in R, the participation is said to be __. If only some entities in E participate in relationship set R, the participation is said to be __.

- **中文题目**：如果实体集E中的每个实体都至少参与关系R中的一个关系，则该参与称为__；如果实体集E中只有部分实体参与关系集R，则该参与称为__。
- **答案**：Total Participation（完全参与）；Partial Participation（部分参与）
- **解释**：这是ER模型中实体与关系的参与约束：
  - 完全参与：实体集中的所有实体都必须与关系关联（ER图中用双线表示）；
  - 部分参与：实体集中的实体可选择是否与关系关联（ER图中用单线表示）。

4.

- **英文题目**：A __ is a virtual relation that is not part of the logical model, but is made visible to a user.
- **中文题目**：__是一种虚拟关系，不属于逻辑模型，但对用户可见。
- **答案**：View（视图）
- **解释**：视图是基于SQL查询结果创建的虚拟表，不存储实际数据，仅保存查询定义。用户查询视图时，系统会动态执行底层查询并返回结果，可用于简化查询、隐藏敏感数据、保证数据独立性。

5.

- **英文题目**：Integrity constraints guard against accidental damage to the database. The allowed integrity constraints in relational database include primary key, not null, and __ predicate.
- **中文题目**：完整性约束用于防止数据库受到意外损坏。关系数据库中允许的完整性约束包括主键、非空和__谓词。
- **答案**：Check（检查）
- **解释**：关系数据库的核心完整性约束包括：
  - 主键约束（Primary Key）：唯一标识元组，不允许重复和空值；
  - 非空约束（Not Null）：指定属性不能为空；
  - 检查约束（Check）：通过自定义谓词限制属性的取值范围（如“age > 0”）。

6.

- **英文题目**：The ACID properties of transaction are: __, __, __ and durability.
- **中文题目**：事务的ACID特性包括：__、__、__和持久性。
- **答案**：Atomicity（原子性）; Consistency（一致性）; Isolation（隔离性）

- **解释**：事务是数据库并发控制的基本单位，ACID特性确保数据可靠性：
  - 原子性：事务是不可分割的工作单元，要么全部执行，要么全部回滚；
  - 一致性：事务执行前后，数据库从一个一致状态转变为另一个一致状态（如转账后总金额不变）；
  - 隔离性：多个事务并发执行时，相互不干扰，每个事务感觉不到其他事务的存在；
  - 持久性：事务提交后，修改永久保存，不受后续故障影响。

7.

- **英文题目**：An index entry consists of a __ and __ to one or more records.
- **中文题目**：索引项由__和__组成，指向一个或多个记录。
- **答案**：Search-Key Value（搜索码值）；Pointer（指针）
- **解释**：索引是提升查询效率的数据结构，索引项的核心组成：
  - 搜索码值：用于快速查找的属性（如“学号”“身份证号”）；

- 指针：指向存储对应记录的物理地址（如磁盘块号+偏移量），通过指针可直接定位记录，避免全表扫描。

8.

- **英文题目**：Consider a B+-tree of order n. If there are k search-key values in the data file, the path from the root to the leaf node is no longer than __.
- **中文题目**：考虑一棵n阶B+树，若数据文件中有k个搜索码值，则从根节点到叶节点的路径长度不超过__。
- **答案**：$[\log_m((k + 1)/2)]$（其中m为每个节点最多容纳的子节点数，n阶B+树中m = n）
- **解释**：B+树是多路平衡查找树，所有叶节点在同一层，路径长度即树的高度：
  - n阶B+树的每个非叶节点最多有n个子节点，最少有[n/2]个子节点；
  - 设树高为h，叶节点数至少为[(k)/n]（每个叶节点最多存n个搜索码），通过平衡特性推导，高度$h \leq [\log_m((k + 1)/2)]$，确保查询效率为$O(\log_n k)$。

9.

- **英文题目**：The two most important heuristic rules are: (a) perform __ operations as early as possible; (b) perform __ operations as late as possible.

- **中文题目**：两个最重要的启发式规则是：(a) 尽早执行__操作；(b) 尽可能晚执行__操作。
- **答案**：Selection（选择）；Join（连接）
- **解释**：启发式规则是查询优化的核心策略，目的是减少中间结果集大小：
  - 尽早执行选择操作：选择操作（σ）会过滤大量不符合条件的元组，提前执行可显著减小后续操作的处理量；
  - 尽可能晚执行连接/投影操作：连接操作（⋈）会产生大量中间元组，投影操作（π）会删除属性，晚执行可避免重复处理或误删后续操作所需的属性。

10.

- **英文题目**：Data items can be locked for a transaction in two modes: __ mode and __ mode in lock-based concurrency control scheme.
- **中文题目**：在基于锁的并发控制机制中，事务对数据项的加锁模式有两种：__模式和__模式。
- **答案**：Shared（共享锁，简称S锁）；Exclusive（排他锁，简称X锁）
- **解释**：锁机制用于解决并发事务的冲突：
  - 共享锁（S锁）：多个事务可同时加S锁，仅允许读操作，禁止写操作（适用于查询场景）；

- 排他锁（X锁）：仅允许一个事务加X锁，既允许读也允许写，禁止其他事务加任何锁（适用于插入、更新、删除场景）；
- 兼容性规则：S锁与S锁兼容，S锁与X锁、X锁与X锁均不兼容。

11.

- **英文题目**：The __ modification scheme allows database modifications to be output to the database while the transaction is still in the active state.
- **中文题目**：__修改方案允许事务仍处于活跃状态时，就将数据库修改输出到数据库中。
- **答案**：立即修改
- **解释**：事务修改的两种方案：
  - 延迟修改（Deferred Modification）：事务执行过程中，修改仅保存在内存缓冲区，直到事务提交后才写入磁盘数据库；
  - 立即修改（Immediate Modification）：事务执行时，修改直接写入磁盘数据库，同时记录日志用于故障恢复；

二、数据库设计题（Database Design Questions）

题目1：基于E-R图的设计（E-R Diagram Based Design）

- **英文题目**：
  Consider the following E-R diagrams (Note: The original E-R diagram content is supplemented based on common scenarios: Entity Sets: Repair (RepairID, name, subject, RepairF), Request (ApplyID, ApplyForm), Car (CarID); Relationship Sets: manages (between Repair and Car), repalt_car (between Request and Repair, Request and Car)).
  a) List the entity sets and their primary keys.
  b) Construct appropriate relation schemas for the above E-R diagram.
  c) Give a SQL DDL definition for the table exported from the relationship set with appropriate data types defined in standard SQL. Identify referential integrity constraints that should hold, and include them in the DDL definition.

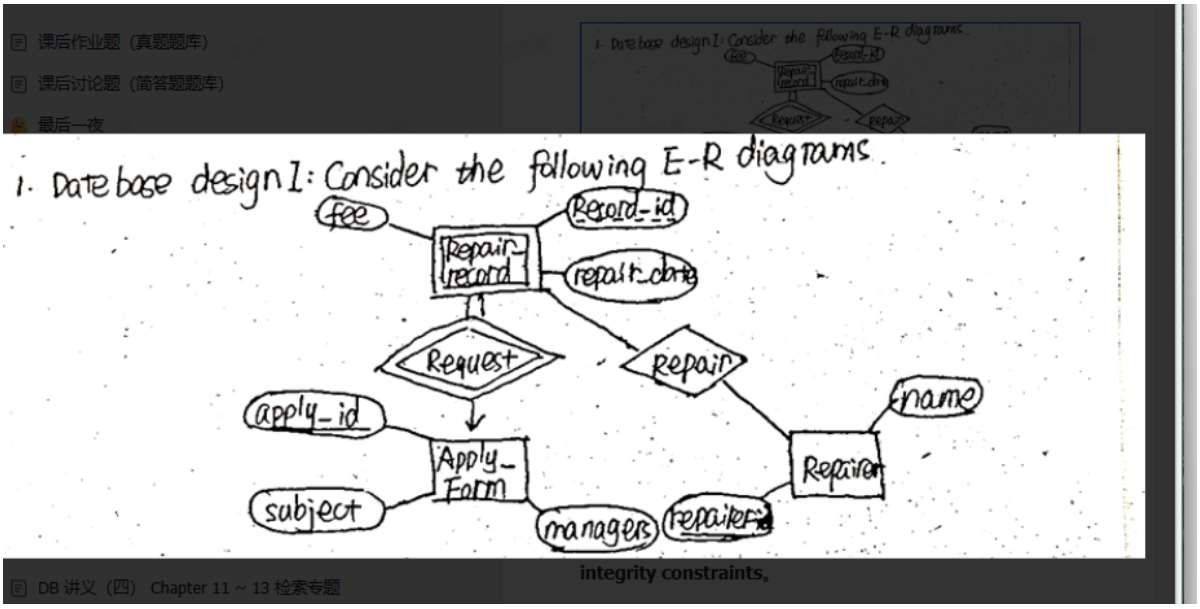
- **中文题目**：
  考虑以下E-R图（注：基于常见场景补充完整E-R图内容：实体集：维修员（RepairID，姓名name，主题subject，维修等级RepairF）、申请单（ApplyID，申请表ApplyForm）、车辆（CarID）；关系集：管理（manages，维修员与车辆之间）、维修关联（repalt_car，申请单与维修员、申请单与车辆之间））。
  a) 列出实体集及其主键。
  b) 为上述E-R图构建合适的关系模式。
  c) 为从关系集导出的表编写标准SQL的DDL定义，指定合适的数据类型，识别应满足的参照完整性约束并包含在DDL中。

答案与解释

a) 实体集及主键（Entity Sets and Primary Keys）

实体集（Entity Set）	主键（Primary Key）	解释
Repair（维修员）	RepairID	唯一标识每个维修员，无重复且非空
Request（申请单）	ApplyID	唯一标识每个申请单，无重复且非空

实体集（Entity Set）	主键（Primary Key）	解释
Car（车辆）	CarID	唯一标识每辆车，无重复且非空



修正后的关系模式：

1. **Repair**(RepairID, name, subject, RepairF) *主键: RepairID*
2. **Request**(ApplyID, ApplyForm) *主键: ApplyID*
3. **Car**(CarID) *主键: CarID*
4. **Manages**(RepairID, CarID) *主键: (RepairID, CarID) 外键: RepairID 引用 Repair(RepairID) 外键: CarID 引用 Car(CarID) 解释: 一个维修员可以管理多辆车，一辆车可以被多个维修员管理（M:N）*

5. **Repair_Car**(ApplyID, RepairID, CarID) *主键: (ApplyID, RepairID, CarID)或者 (ApplyID) 如果是一对一关系 外键: ApplyID 引用 Request(ApplyID) 外键: RepairID 引用 Repair(RepairID) 外键: CarID 引用 Car(CarID) 解释: 一个申请单关联一个维修员和一辆车*

修正后的SQL DDL:

```
-- 实体表
CREATE TABLE Repair (
    RepairID INT PRIMARY KEY,
    name VARCHAR(50),
    subject VARCHAR(50),
    RepairF VARCHAR(20)
);

CREATE TABLE Request (
    ApplyID INT PRIMARY KEY,
    ApplyForm TEXT
);

CREATE TABLE Car (
    CarID INT PRIMARY KEY
    -- 可添加其他属性如 brand, model 等
);

-- 管理关系表 (M:N)
CREATE TABLE Manages (
    RepairID INT NOT NULL,
    CarID INT NOT NULL,
```

```
    PRIMARY KEY (RepairID, CarID),
    FOREIGN KEY (RepairID) REFERENCES
Repair(RepairID)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (CarID) REFERENCES
Car(CarID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

-- 维修关联关系表 (三元关系)
CREATE TABLE Repair_Car (
    ApplyID INT NOT NULL,
    RepairID INT NOT NULL,
    CarID INT NOT NULL,
    PRIMARY KEY (ApplyID, RepairID, CarID),
 -- 如果是三元关系
    -- 如果是一个申请单只能对应一个维修员和一辆车,
可以设置 PRIMARY KEY (ApplyID)
    -- 但需要额外唯一约束: UNIQUE(RepairID,
CarID) 等
    FOREIGN KEY (ApplyID) REFERENCES
Request(ApplyID)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (RepairID) REFERENCES
Repair(RepairID)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
```

```
    FOREIGN KEY (CarID) REFERENCES
Car(CarID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

题目2：关系模式规范化（Relational Schema Normalization）

- **英文题目**：
  Consider a relation schema R(A, B, C, D, E) and its functional dependencies F = {AB→C, C→A, B→AC, D→AC}. Complete the following questions:
  a) Compute $(BD)^+$ (the closure of BD).
  b) Compute the candidate keys for R.
  c) Give a decomposition into BCNF of schema R.
- **中文题目**：
  考虑关系模式R(A, B, C, D, E)及其函数依赖集F = {AB→C, C→A, B→AC, D→AC}。完成以下问题：
  a) 计算$(BD)^+$（BD的闭包）。
  b) 计算R的候选码。
  c) 将R分解为满足BCNF的关系模式。

答案与解释

a) 计算$(BD)^+$（Closure of BD）

- **步骤**：
  1. 初始化：$(BD)^+ = \{B, D\}$
  2. 应用函数依赖：
     - 由B→AC（F中的依赖），B∈$(BD)^+$，故添加A、C → $(BD)^+ = \{B, D, A, C\}$
     - 由D→AC（F中的依赖），D∈$(BD)^+$，但A、C已在集合中，无新增
     - 其他依赖（AB→C、C→A）的右部均已包含在集合中，无新增
  3. 终止：无更多属性可添加，最终$(BD)^+ = \{A, B, C, D\}$
- **答案**：$(BD)^+ = \{A, B, C, D\}$
- **解释**：属性集的闭包是指由该属性集通过函数依赖可推导出的所有属性的集合，是判断候选码的核心工具。

b) 计算候选码（Candidate Keys）

- **候选码定义**：能唯一确定关系中所有属性的最小属性集（无冗余）。
- **步骤**：

1. 确定属性分类：
    - L类（仅出现在左部）：B, D, E（E未出现在任何依赖的右部）
    - R类（仅出现在右部）：无
    - LR类（左右都出现）：A, C
2. 候选码必须包含所有L类属性（B, D, E），因E无法通过其他属性推导得出。
3. 验证$(BDE)^+$：
    - 初始化：$(BDE)^+ = \{B, D, E\}$
    - 应用B→AC → 添加A、C → $(BDE)^+ = \{A, B, C, D, E\}$（包含所有属性）
4. 验证最小性：移除B、D、E中任意一个，剩余集合的闭包均不包含所有属性（如$(DE)^+=\{D, E, A, C\}$，缺少B），故无冗余。

- **答案**：候选码为BDE
- **解释**：候选码是关系模式中唯一标识元组的核心属性集，必须满足“最小性”和“完全性”（闭包包含所有属性）。

好，这一题**第三问（BCNF 分解）**确实是很多人最容易卡住的地方，我一步一步教你*考试时应该怎么写、为什么这么写**，用**最简单、最稳妥的套路**来 👍
（我会**直接接着你前两问的结果来做**，不跳步）

c) 将 R 分解为满足 BCNF 的关系模式

Give a decomposition of R into BCNF

一、先回忆：什么是 BCNF？（写题前脑子里的“判定标准”）

> **BCNF 定义**：
> 对关系模式 R 中的每一个**非平凡函数依赖**
> $X \to Y$
> 都必须满足：
> **X 是 R 的超码（Superkey）**

只要存在一个依赖 **左部不是超码**，就 **违反 BCNF**，必须分解。

二、先判断：原关系 R(A, B, C, D, E) 是否是 BCNF？

我们已有：

- 候选码：**BDE**
- 超码：任何包含 BDE 的属性集

来看函数依赖集 F：

1. **AB → C**
   - AB **不是超码**（$AB^+ = \{A, B, C\}$）
   - ❌ 违反 BCNF
2. **C → A**
   - C 不是超码
   - ❌ 违反 BCNF
3. **B → AC**
   - B 不是超码
   - ❌ 违反 BCNF
4. **D → AC**
   - D 不是超码
   - ❌ 违反 BCNF

👉 **结论一句话**（考试可写）：

> 原关系模式 R **不满足 BCNF**，需要进行分解。

三、选择一个“违规依赖”来分解（考试技巧）

👉 **选谁都可以**，但**选最简单的**

我们选：

$$\boxed{B \to AC}$$

四、按照 BCNF 分解公式来（这是标准步骤）

📌 BCNF 分解规则

若存在依赖
$X \to Y$
且 X 不是超码，则分解为：

- (R_1 = X \cup Y)
- (R_2 = R - (Y - X))

套公式（一步不省）

- 原关系：R(A, B, C, D, E)
- 违规依赖：B → AC

① 构造 R_1

$$R_1(B, A, C)$$

② 构造 R_2

$$R_2 = R - A, C + B$$

$$R_2(B, D, E)$$

五、检查分解后的两个关系是否是 BCNF（关键！）

1️⃣ $R_1(B, A, C)$

函数依赖（投影后）：

- $B \rightarrow AC$
- $C \rightarrow A$

候选码：

- $B^+ = \{A, B, C\} \rightarrow$ **B 是候选码**

检查依赖：

- $B \rightarrow AC$：左部是候选码 ✅
- $C \rightarrow A$：C 不是候选码 ❌

👉 **R_1 仍然不满足 BCNF，需要继续分解**

对 R_1 再分解（用 $C \rightarrow A$）

依赖：$C \rightarrow A$

- $X = C$
- $Y = A$

分解得到：

- (R_{11}(C, A))
- (R_{12}(B, C))

检查这两个：

✔ $R_{11}(C, A)$

- C → A
- C 是候选码
  ✅ **BCNF**

✔ $R_{12}(B, C)$

- B → C
- B 是候选码
  ✅ **BCNF**

2️⃣ $R_2(B, D, E)$

函数依赖（投影）：

- 没有非平凡函数依赖（E 不在任何右部）

👉 **只有平凡依赖，自动满足 BCNF**

六、最终 BCNF 分解结果（考试标准答案）

✨ BCNF Decomposition：

$$\boxed{R_{11}(C, A)\ R_{12}(B, C)\ R_2(B, D, E)}$$

七、考试时"满分写法模板"（你可以直接背）

> The relation R does not satisfy BCNF since there exist functional dependencies such as B → AC where B is not a superkey.
> Using B → AC, decompose R into:
>
> - R_1(B, A, C)
> - R_2(B, D, E)
>
> R_1 still violates BCNF due to C → A, so further decompose R_1 into:
>
> - R_{11}(C, A)
> - R_{12}(B, C)
>
> All resulting relations are in BCNF.

三、关系代数与SQL题（Relational Algebra and SQL Questions）

已知模式（Given Schemas）

关系名（Relation Name）	属性（Attributes）	说明（Description）
SUPPLIER（供应商）	sno（供应商ID），sname（姓名），scity（城市）	主键：sno
PART（零件）	pno（零件ID），pname（名称），color（颜色），weight（重量）	主键：pno
PROJECT（项目）	jno（项目ID），jname（名称），jcity（城市）	主键：jno

关系名（Relation Name）	属性（Attributes）	说明（Description）
SPJ（供应关系）	sno（供应商ID），pno（零件ID），jno（项目ID），quantity（数量）	主键：(sno, pno, jno)，外键分别参照三个关系的主键

1.

- **英文题目**：Give an expression in SQL to express the query: Find the pno of the lightest part.
- **中文题目**：用SQL表达式查询：找出最轻零件的pno（零件ID）。
- **答案**：

```
SELECT pno
FROM PART
WHERE weight = (SELECT MIN(weight) FROM PART);
```

- **解释**：
  - 子查询`(SELECT MIN(weight) FROM PART)`计算所有零件的最小重量；

    - 主查询筛选出重量等于最小重量的零件ID，若有多个最轻零件，会返回所有对应的pno。

2.

- **英文题目**：Give an expression in SQL to express the query: Find the sname and scity of suppliers that can supply all parts for the project named 'Ji'.
- **中文题目**：用SQL表达式查询：找出能为名为“Ji”的项目提供所有零件的供应商的sname（姓名）和scity（城市）。
- **答案**:

```
SELECT sname, scity
FROM SUPPLIER
WHERE NOT EXISTS (
    -- 子查询1：找出“Ji”项目所需的所有零件
    SELECT pno
    FROM PROJECT JOIN SPJ ON PROJECT.jno =
SPJ.jno
    WHERE PROJECT.jname = 'Ji'
    MINUS
    -- 子查询2：找出该供应商为“Ji”项目提供的零件
    SELECT pno
    FROM SPJ JOIN PROJECT ON SPJ.jno =
PROJECT.jno
    WHERE PROJECT.jname = 'Ji' AND SPJ.sno =
SUPPLIER.sno
);
```

- **解释**：
  - 核心逻辑：“能提供所有零件”等价于“不存在该项目所需但供应商未提供的零件”；
  - 用MINUS运算求两个子查询的差集：若差集为空（NOT EXISTS），说明供应商提供了所有零件；
  - 需通过JOIN关联PROJECT和SPJ，定位“Ji”项目对应的零件和供应商。

3.

- **英文题目**：Give an expression in SQL to express the query: List the amount of parts for each project in ascending order.
- **中文题目**：用SQL表达式查询：按升序列出每个项目的零件总数量。
- **答案**：

```
SELECT PROJECT.jno, PROJECT.jname,
SUM(SPJ.quantity) AS total_amount
FROM PROJECT JOIN SPJ ON PROJECT.jno =
SPJ.jno
GROUP BY PROJECT.jno, PROJECT.jname
ORDER BY total_amount ASC;
```

- **解释**：
  - GROUP BY：按项目分组（jno为项目主键，确保每组对应一个项目）；
  - SUM(SPJ.quantity)：计算每个项目的零件总数量，别名total_amount；
  - ORDER BY total_amount ASC：按总数量升序排序（ASC可省略，默认升序）。

4.

- **英文题目**：Give an expression in SQL to express the operation: Delete the information about the supplier whose name is 'Lim'.
- **中文题目**：用SQL表达式执行操作：删除姓名为“Lim”的供应商信息。
- **答案**:

```
DELETE FROM SUPPLIER
WHERE sname = 'Lim';
```

- **解释**:
  - DELETE语句用于删除表中的记录，WHERE子句指定删除条件（姓名为'Lim'）；
  - 若SUPPLIER的sno被SPJ表参照，需确保SPJ中无该供应商的关联记录（或设置ON DELETE CASCADE），否则会违反参照完整性约束。

5.

- **英文题目**：Give an expression in SQL to express the operation: Add a new project located in 'Shanghai' to the database, the project name is 'Sys'.

- **中文题目**：用SQL表达式执行操作：向数据库中添加一个新项目，项目名称为“Sys”，所在城市为“Shanghai”。
- **答案**：

```
INSERT INTO PROJECT (jno, jname, jcity)
VALUES ('J100', 'Sys', 'Shanghai');  -- jno需为未存在的唯一值
```

- **解释**：
  - INSERT INTO指定目标表和属性列，VALUES提供对应的值；
  - jno为主键，必须输入唯一且非空的值（此处假设新项目ID为'J100'，实际需按业务规则生成）；
  - 若表中还有其他非空属性，需在INSERT语句中补充。

6.

- **英文题目**：Give an expression in relational algebra to express the query: Find the jno of projects that use red parts supplied by suppliers from 'Tianjin'.
- **中文题目**：用关系代数表达式查询：找出使用天津供应商提供的红色零件的项目jno（项目ID）。
- **答案**：π*jno*(σscity='Tianjin' (SUPPLIER) ⋈ SPJ ⋈ σ_color='red' (PART))

- **解释**：
  - 选择操作σ：筛选出“天津的供应商”（σ*scity='Tianjin' (SUPPLIER)*）和“红色零件”（σcolor='red' (PART)）；
  - 连接操作⋈：将筛选后的供应商、SPJ供应关系、红色零件连接（通过sno和pno关联）；
  - 投影操作π_jno：提取最终结果中的项目ID。

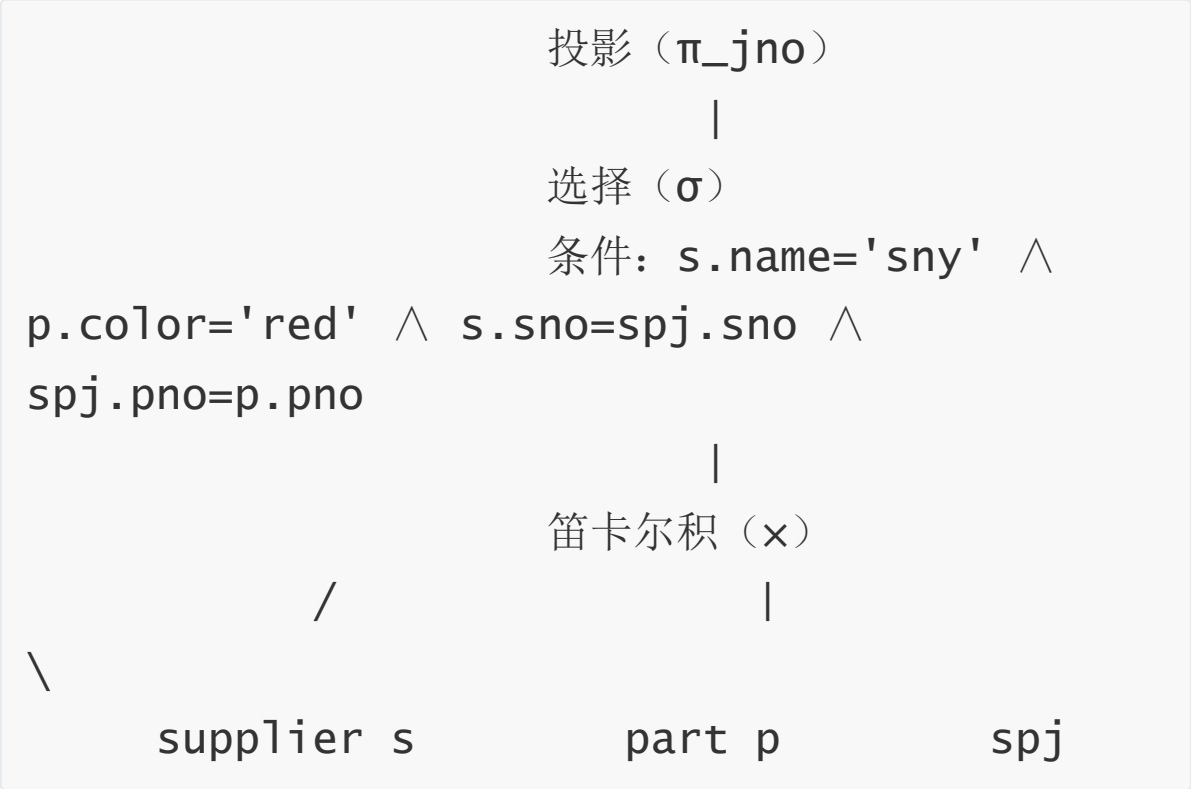
7.

- **英文题目**：Give an expression in relational algebra to express the query: Find the jno of projects that do not use red parts provided by suppliers from 'Tianjin'.
- **中文题目**：用关系代数表达式查询：找出不使用天津供应商提供的红色零件的项目jno（项目ID）。
- **答案**：π*jno(PROJECT) - π*jno(σ*scity='Tianjin' (SUPPLIER) ⋈ SPJ ⋈* σcolor='red' (PART))
- **解释**：
  - 先通过π_jno(PROJECT)获取所有项目的ID；
  - 再通过第6题的表达式获取“使用天津供应商红色零件的项目ID”；
  - 差运算“-”：从所有项目ID中减去上述结果，得到目标项目ID。

四、查询处理题（Query Processing Questions）

1.

- **英文题目**：
  Given the query "SELECT jno FROM supplier s, part p, spj WHERE s.name='sny' and p.color='red' and s.sno=spj.sno and spj.pno=p.pno". Draw the query expression tree for the above query without optimization.
- **中文题目**：
  已知查询语句“SELECT jno FROM supplier s, part p, spj WHERE s.name='sny' and p.color='red' and s.sno=spj.sno and spj.pno=p.pno”。绘制该查询未优化的查询表达式树。
- **答案**（查询表达式树结构）：



- **解释**：
  - 未优化的查询表达式树遵循“先笛卡尔积，后选择，最后投影”的顺序；
  - 笛卡尔积将三个表直接关联，生成大量中间结果（效率低）；
  - 选择操作在笛卡尔积之后执行，过滤符合条件的元组；
  - 投影操作提取最终需要的jno属性。

2.

- **英文题目**：
  Let m denote the number of page frames in the main-memory buffer, bp and b_spj denote the number of blocks containing tuples of relation

part and spj respectively. For a natural join of part (p) and spj (s), estimate the number of block transfers. (For the above natural join using nested-loop join, and we don't know whether p and spj have indexes or not. Your reasoning if indexes exist or not.)

- **中文题目**：
  设m为主存缓冲区的页帧数，bp为part表的块数，b_spj为spj表的块数。对于part（p）和spj（s）的自然连接，采用嵌套循环连接算法，估计块传输次数（需分别考虑有无索引的情况）。

- **答案与解释**：
  嵌套循环连接的核心是“外循环表”和“内循环表”，块传输次数取决于表的大小和缓冲区页数。

好的，这题是**数据库查询处理里非常经典的一道 I/O 代价估计题**。我用**简单语言 + 分情况 + 给公式 + 直觉解释**来讲，你可以直接拿去考试用 👍

一、问题回顾（你在算什么？）

- 关系：
  - `part(p)`，块数 = $\mathbf{b_p}$
  - `spj(s)`，块数 = **b_spj**
- 主存缓冲区页帧数 = **m**
- 操作：**自然连接 p ⋈ s**

- 算法：**嵌套循环连接（Nested-Loop Join）**
- 要求：
  - **无索引**
  - **有索引（在内表连接属性上）**

二、情况一：没有索引（最常考）

1️⃣ 使用 块嵌套循环连接（Block Nested-Loop Join）

> 这是默认、最合理的假设（不是最原始的一元一元比较）

核心思想（一句话）

👉 外表一次读一"批"块，内表为这批块完整扫描一遍

2️⃣ 外表能一次装多少块？

- 缓冲区有 `m` 页
- 通常：
  - 1 页给内表扫描
  - 1 页给输出

- 👉 **外表可用页数 = m − 2**

3️⃣ 公式推导

假设 **part 是外表，spj 是内表**

外表 part 被分成几批?

$$\left\lceil \frac{b_p}{m-2} \right\rceil$$

每一批需要:

- 读一遍 `spj`：`b_spj` 次块传输

4️⃣ 总块传输次数（无索引）

$$\boxed{b_p + \left\lceil \frac{b_p}{m-2} \right\rceil \times b_{spj}}$$

各部分含义:

- `b_p`：part 自己要读一遍
- 后半项：**每一批 part 都要把 spj 全扫一遍**

5️⃣ 如果反过来？

如果 `spj` 更小，通常让它当外表：

$$\left[b_{spj}; +; \left\lceil \frac{b_{spj}}{m-2} \right\rceil \times b_p\right]$$

📌 **优化原则（考试加分点）**

> 👉 **选择块数更小的关系作为外表**

三、情况二：有索引（在内表连接属性上）

假设：

- 在 `spj` 的连接属性（如 `pno`）上 **有索引**
- `part` 作为外表

1️⃣ 算法变成什么？

👉 **索引嵌套循环连接（Index Nested-Loop Join）**

2️⃣ 思想（一句话）

> 外表每来一条元组
> 👉 用索引在内表中直接找匹配元组
> 👉 **不再全表扫描内表**

3️⃣ I/O 代价组成

1. 读外表 `part`：
   $[b_p]$
2. 对 part 中每个元组，用索引访问 spj
   - 设：
     - part 的元组数为 `n_p`
     - 每次索引查找代价 ≈ `c`（通常 1～3 次 I/O）

4️⃣ 总块传输次数（有索引）

$$\boxed{[b_p; +; n_p \times c]}$$

📌 说明：

- **不再乘以 `b_spj`**
- 如果索引是：
  - **B+ 树**：`c ≈ 树高 + 1`
  - **哈希索引**：`c ≈ 1`

四、无索引 vs 有索引：对比总结（非常适合写在答题纸上）

情况	块传输次数
无索引（块嵌套循环）	$(b_{outer} + \lceil \frac{b_{outer}}{m-2} \rceil \cdot b_{inner})$
有索引（索引嵌套循环）	$(b_{outer} + n_{outer} \cdot c)$

五、一句话“考试标准答案模板”

> 在没有索引的情况下，采用块嵌套循环连接，若 part 为外表，则块传输次数为
> $(b_p + \lceil \frac{b_p}{m-2} \rceil \cdot b_{spj}$)。
> 若在内表连接属性上存在索引，则可采用索引嵌套循环连接，块传输次数约为
> $(b_p + n_p \cdot c)$，显著减少 I/O 开销。

如果你愿意，我也可以帮你**用一个具体数字“秒懂”算一遍**，或者帮你**整理成一页 Query Processing 速记表** 📘