2014数据库MediUm

以下是基于现有信息整理的RDBMS（关系型数据库管理系统）相关选择题，包含中英文问题、答案及详细解释，部分题目结合数据库核心知识点补充说明：

2. 题目

- **英文**：Let R be a relation schema, R1 and R2 form a decomposition of R. Decomposition is a __ if for all legal databases instances r of R, the natural join of R1 and R2 (from r) equals r.
- **中文**：设R为关系模式，R1和R2构成R的一个分解。若对R的所有合法数据库实例r，R1和R2（从r导出）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核心要求之一，其本质是“分解后不丢失原关系的信息”。通过自然连接可完全恢复原关系r，说明分解过程中没有产生冗余的“伪元组”，也没有丢失原有的数据关联。反之，若分解为“有损分解”，自然连接的结果会包含原关系中不存在的伪元组，导致信息丢失。

5. 题目

- **英文**：For evaluating an entire expression tree, __ passes on tuples to parent operations even as an operation is being executed.
- **中文**：在评估整个表达式树时，__ 在某个操作执行过程中就将元组传递给父操作。
- **答案**：producer-driven (push/eager)（生产者驱动/推送式/主动式）
- **解释**：
  数据库查询执行中，表达式树的评估有两种核心方式：
  - 生产者驱动（push）：操作（如选择、投影、连接）一旦生成部分元组，立即将其“推送”给父操作，无需等待整个操作执行完毕。这种方式响应快，可并行处理部分结果，适合流式数据场景。
  - 消费者驱动（pull/lazy）：父操作主动向子操作“拉取”元组，子操作仅在收到拉取请求时才生成元组。
    题干中“执行中传递元组”的描述符合生产者驱动的特征，故答案为producer-driven（括号内为同义表述，push强调推送动作，eager强调主动执行）。

6. 题目

- **英文**：The time it takes for a disk I/O includes block-transfer time and __ time. The latter consists of __ time and rotational latency.
- **中文**：磁盘I/O的时间包括块传输时间和**时间。后者由**时间和旋转延迟组成。
- **答案**：disk seek time（寻道时间）；seek（寻道）
  （注：原文档中“MTTF”为干扰项，MTTF是“平均无故障时间”，与磁盘I/O时间构成无关，此处修正为正确逻辑）
- **解释**：
  磁盘I/O总时间是数据库性能优化的关键指标，其构成如下：
  1. 寻道时间（seek time）：磁头移动到目标磁道所需的时间，是磁盘I/O中耗时最长的部分。
  2. 旋转延迟（rotational latency）：目标扇区旋转到磁头下方所需的时间，取决于磁盘转速（如7200转/分的磁盘平均旋转延迟约4ms）。
  3. 块传输时间（block-transfer time）：将数据从磁盘扇区读取到内存或写入磁盘的时间，取决于数据块大小和磁盘传输速率。
     题干中“后者”指前一空的答案（寻道时间+旋转延迟），因此两空依次为“disk seek time”和“seek time”（第二空简化为seek，符合题干语法）。

8. 题目

- **英文**：A sequence of primitive operations that can be used to evaluate a query is a __ plan.
- **中文**：可用于评估查询的一系列基本操作的集合称为__计划。
- **答案**：query execution plan（查询执行计划）或 execution plan（执行计划）
- **解释**：
  数据库查询处理分为三个阶段：查询解析（语法分析）、查询优化（选择最优策略）、查询执行。查询执行计划是优化后的具体执行方案，包含一系列 primitive operations（基本操作，如扫描、连接、排序、聚合等），明确了操作的执行顺序、数据访问方式（如全表扫描、索引扫描）、连接算法（如嵌套循环连接、哈希连接）等。数据库引擎通过执行该计划完成查询结果的计算。

10. 题目

- **英文**：In query processing, the query-execution engine takes the __ plan which contains detailed information on how a particular query or a set of queries will be executed.
- **中文**：在查询处理中，查询执行引擎采用__计划，该计划包含特定查询或一组查询的详细执行方式信息。

- **答案**：detailed query execution plan（详细查询执行计划）或query execution plan（查询执行计划）
- **解释**：
  该题与第8题互为补充，强调“包含详细执行信息”的核心特征。查询执行计划分为逻辑计划（仅描述操作逻辑，如“连接R1和R2”）和物理计划（包含详细执行细节，如“用哈希连接算法连接R1和R2，使用R1的索引扫描”）。题干中“detailed information on how to be executed”明确指向物理层面的查询执行计划，其作用是为查询执行引擎提供可直接执行的步骤。

11. 题目

- **英文**：In deferred database modification scheme, __ operation is the only operation used in the recovery procedure.
- **中文**：在延迟数据库修改方案中，__ 操作是恢复过程中唯一使用的操作。
- **答案**：Redo（重做）（原文档“commit”为错误，修正如下）
- **解释**：
  延迟数据库修改（deferred modification）是数据库恢复技术中的重要方案，其核心机制是：
  - 事务执行时，所有修改操作仅记录在日志（log）中，不立即写入数据库（磁盘）。

- 只有当事务提交（commit）时，才将日志中的修改批量写入数据库。
- 若事务执行过程中发生故障，由于未对数据库做实际修改，无需执行“Undo（撤销）”操作；恢复时仅需对已提交但未写入数据库的事务执行“Redo（重做）”，将日志中的修改重新应用到数据库。
  因此，延迟修改方案的恢复过程仅需Redo操作，commit是事务的提交动作，并非恢复操作，故原答案修正为Redo。

12. 题目

- **英文**：Cascading rollbacks can be avoided by applying __ two phase locking protocol to transactions in concurrency protocols.
- **中文**：在并发控制协议中，通过对事务应用__两段锁协议，可以避免级联回滚。
- **答案**：strict（严格两段锁协议）或Rigorous（严谨两段锁协议）
- **解释**：
  两段锁协议（2PL）是保证事务串行化的基础协议，但普通2PL可能导致级联回滚（cascading rollbacks）：若事务T1修改了数据并持有锁，事务T2读取了该未提交的数据（脏读），之后T1回滚，T2也必须回滚，进而可能引发更多依赖T2的事务回滚，形

成级联效应。
为避免级联回滚，需对2PL进行增强：

- 严格两段锁协议（Strict 2PL）：事务提交后才释放排他锁（写锁），共享锁（读锁）可在查询结束后释放。确保其他事务只能读取已提交的数据，避免脏读，从而杜绝级联回滚。
- 严谨两段锁协议（Rigorous 2PL）：事务提交后才释放所有锁（共享锁+排他锁），约束更强，同样能避免级联回滚。
  两者均能满足题干要求，故答案为strict或Rigorous。

13. 题目

- **英文**：__ is the most widely used structure for recording the modification of database. Since a failure may occur while an update is taking place, it must be written out to __ storage before the actual update to database to be done.
- **中文**：__ 是记录数据库修改最广泛使用的结构。由于更新过程中可能发生故障，必须在实际修改数据库之前将其写入__存储。
- **答案**：Log（日志）；permanent（永久）

- **解释**：
  数据库日志（Log）是恢复机制的核心，用于记录所有事务对数据库的修改操作（如插入、删除、更新），包含操作类型、数据旧值、新值、事务ID等关键信息。
  为保证故障恢复的可靠性，日志必须遵循“先写日志，后写数据库”（Write-Ahead Logging, WAL）原则：在将修改写入数据库（易失性存储，如内存缓存）之前，必须先将对应的日志记录写入永久存储（如磁盘）。这样即使更新过程中发生故障（如断电），可通过日志中的记录执行Redo或Undo操作，恢复数据库一致性。永久存储的特征是断电后数据不丢失，符合日志的可靠性要求。

14. 题目

- **英文**：A (possibly concurrent) schedule is serializable if it is __ to a serial schedule.
- **中文**：一个（可能是并发的）调度是可串行化的，当且仅当它与某个串行调度__。
- **答案**：equivalent（等价）
- **解释**：
  并发事务调度的核心目标是“既提高效率，又保证数据一致性”，可串行化是衡量并发调度正确性的标准：
  - 串行调度：多个事务按顺序执行，无并发冲突，一定是正确的，但效率低。

- 并发调度：多个事务交替执行，效率高，但可能产生冲突（如脏读、不可重复读）。
  若一个并发调度的执行结果与某个串行调度（事务执行顺序不同，但最终数据状态一致）等价，则称其为可串行化调度。等价性通常通过“冲突等价”（Conflicting Equivalence）或“视图等价”（View Equivalence）定义，核心是确保并发执行不改变串行执行的最终结果，从而保证数据一致性。

2. 数据库设计：关系模式 $R(A, B, C, D, E)$ 的范式分析

（对应题目：Given a relation schema $R(A, B, C, D, E)$ with functional dependencies $ABC \to D$, $D \to E$, $E \to A$.）

a) 找出 R 的所有候选码

英文问题：Find all candidate keys for R.
中文问题：找出 R 的所有候选码。

答案：候选码为 ABC、BCD、BCE。

解释：
候选码是能**唯一确定关系中所有属性**的最小属性集，需通过**属性闭包**计算：

- 计算 $(ABC)^+$：
  由 $ABC \to D$，得 $(ABC)^+ = \{A, B, C, D\}$；
  再由 $D \to E$，得 $(ABC)^+ = \{A, B, C, D, E\}$（覆盖所有属性），故 ABC 是候选码。
- 计算 $(BCD)^+$：
  由 $D \to E$，得 $(BCD)^+ = \{B, C, D, E\}$；
  再由 $E \to A$，得 $(BCD)^+ = \{A, B, C, D, E\}$，故 BCD 是候选码。
- 计算 $(BCE)^+$：
  由 $E \to A$，得 $(BCE)^+ = \{B, C, E, A\}$；
  再由 $ABC \to D$（已包含 A, B, C），得 $(BCE)^+ = \{A, B, C, D, E\}$，故 BCE 是候选码。

b) 将 R 分解为 3NF

英文问题：Decompose R into 3NF (only decompose if there is a violation of 3NF; if R is already in 3NF, write $R(A, B, C, D, E)$).
中文问题：将 R 分解为3NF（仅当存在3NF违例时分解；若 R 已满足3NF，直接写 $R(A, B, C, D, E)$）。

答案：$R(A, B, C, D, E)$（已满足3NF）。

解释：
3NF的要求是：

1. 关系满足2NF（无**非主属性对候选码的部分依赖**）；
2. 无**非主属性对候选码的传递依赖。**

首先，R 的所有属性（A, B, C, D, E）都是**主属性**（属于至少一个候选码），因此不存在“非主属性”。由于3NF的约束仅针对“非主属性”，故 R 天然满足3NF，无需分解。

c) 将 R 分解为 BCNF

英文问题：Decompose R into BCNF.
中文问题：将 R 分解为BCNF。

答案：分解结果为 $R1(B, C, D)$、$R2(D, E)$、$R3(E, A)$。

解释：
BCNF的要求是：**所有函数依赖的左部都是候选码。**

原关系 R 中，函数依赖 $D \to E$、$E \to A$ 的左部（D、E）不是候选码，违反BCNF，需分解：

1. 针对 $D \to E$：将 R 分解为 $R1(A, B, C, D)$ 和 $R2(D, E)$。
   检查 $R1$：其函数依赖为 $ABC \to D$、$(D \to A)$（由 $D \to E$ 和 $E \to A$ 传递得到）。

其中 $D \to A$ 的左部 D 不是 $R1$ 的候选码（$R1$ 的候选码是 ABC），仍违反BCNF。

2. 针对 $R1$ 中的 $D \to A$：将 $R1$ 分解为 $R11(B, C, D)$ 和 $R12(D, A)$。
   此时 $R11$ 的候选码是 BCD，函数依赖 $BCD \to$ 自身属性，满足BCNF；
   $R12$ 的候选码是 D，函数依赖 $D \to A$ 满足BCNF；
   $R2(D, E)$ 的候选码是 D，函数依赖 $D \to E$ 满足BCNF；
   合并 $R12(D, A)$ 与原 $E \to A$ 对应的 $R3(E, A)$，最终分解为 $R1(B, C, D)$、$R2(D, E)$、$R3(E, A)$。

要不要我帮你整理一份**候选码与范式分解的步骤总结表**？

4. 查询处理：排序-归并连接（Sort-Merge Join）的代价计算

英文问题：Relation R contains 10,000 tuples (10 tuples per page), and relation S contains 2,000 tuples (10 tuples per page). If only 15 buffers are available, what is the cost of a sort-merge join?
中文问题：关系 R 包含10,000个元组（每页10个元组），关系 S 包含2,000个元组（每页10个元组）。若仅15个缓冲区可用，排序-归并连接的代价是多少?

步骤1：计算关系的页数（块数）

首先将元组总数转换为**磁盘块数**（每页对应1个磁盘块）：

- R 的块数 $b_R = \frac{10000}{10} = 1000$
- S 的块数 $b_S = \frac{2000}{10} = 200$

步骤2：排序-归并连接的代价构成

排序-归并连接的代价 = **排序阶段的代价** + **归并连接阶段的代价**。

（1）排序阶段的代价（外部排序）

外部排序的代价公式（基于“归并排序”，缓冲区数 $M=15$）：
总块传输数 = $b \times (2\lceil \log_{M-1}(b/M) \rceil + 1)$
（注：$M-1$ 是每趟归并的输入段数；公式包含“读+写”的块传输，最后一趟仅读不写）

- 对 R 排序的代价：
  初始归并段数 $\lceil b_R/M \rceil = \lceil 1000/15 \rceil = 67$
  归并趟数 $\lceil \log_{14}(67) \rceil = 2$（因为 $14^1 = 14 < 67$，$14^2 = 196 \geq 67$）
  排序代价
  $= 1000 \times (2 \times 2 + 1) = 1000 \times 5 = 5000$ 块传输

- 对 S 排序的代价：
  初始归并段数 $\lceil b_S/M \rceil = \lceil 200/15 \rceil = 14$
  归并趟数 $\lceil \log_{14}(14) \rceil = 1$
  排序代价 $= 200 \times (2 \times 1 + 1) = 200 \times 3 = 600$ 块传输

（2）归并连接阶段的代价

排序完成后，归并连接的代价是**读取两个已排序关系的块数**（无需重复读取）：
归并连接代价 $= b_R + b_S = 1000 + 200 = 1200$ 块传输

步骤3：总代价（块传输数）

总代价 = 排序 R 的代价 + 排序 S 的代价 + 归并连接的代价
$= 5000 + 600 + 1200 = 6800$ 块传输

补充：若包含磁盘搜索代价

若考虑磁盘搜索（假设每个归并段分配 1 个缓冲区 $b_b = 1$）：

- 排序 R 的搜索代价：
  $2\lceil 1000/15 \rceil + \lceil 1000/1 \rceil \times (2 \times 2 - 1) = 2 \times 67 + 1000 \times 3 = 3134$
- 排序 S 的搜索代价：
  $2\lceil 200/15 \rceil + \lceil 200/1 \rceil \times (2 \times 1 - 1) = 2 \times 14 + 200 \times 1 = 228$
- 归并连接的搜索代价：$\lceil 1000/15 \rceil + \lceil 200/15 \rceil = 67 + 14 = 81$

  总搜索代价 = $3134 + 228 + 81 = 3443$

答案：排序 - 归并连接的**块传输代价为 6800**（若含搜索代价则需额外计算）。

要不要我帮你整理一份**排序 - 归并连接代价计算的步骤模板**？

我会把每一步拆解得更细，用“大白话+公式”的方式解释~

先明确几个基础概念

在数据库中，“**代价**”通常指“磁盘I/O的块传输次数”（因为磁盘读写是查询中最慢的环节）。

- **块（Page）**：数据库中数据的存储单位，题目里“每页10个元组”→ 1个块=10个元组。
- **缓冲区（Buffer）**：内存中临时存放磁盘块的区域，题目里“15个缓冲区”→ 内存最多同时存15个块。
- **排序-归并连接（Sort-Merge Join）**：分两步：①先把两个关系按“连接属性”排序；②再把排序后的关系“归并”（类似合并两个有序数组）。

步骤1：计算关系的“块数”（先把元组转成块）

关系的总块数 = 总元组 ÷ 每页元组

- 关系R：总元组10000，每页10个 → 块数 $b_R = 10000 \div 10 = 1000$ 块
- 关系S：总元组2000，每页10个 → 块数 $b_S = 2000 \div 10 = 200$ 块

步骤2：理解“排序阶段的代价”（外部排序，因为数据太大内存装不下）

内存只有15个缓冲区（最多存15块），但R有1000块、S有200块，必须用“外部排序”（分多趟把数据读入内存排序，再写回磁盘）。

外部排序的核心逻辑是“**分治**”：

1. **拆分阶段**：把大关系拆成多个“小片段”（叫“归并段”），每个片段能放进内存（15块）。
2. **归并阶段**：把多个已排序的小片段，逐步合并成一个完整的有序关系。

外部排序的代价公式（块传输数）

总块传输数 = 块数 $\times(2\times$归并趟数 $+1)$

- 解释：每一趟排序需要“读一次块 + 写一次块”（除了最后一趟只需要读，不用写），所以每趟的代价是“2×块数”，最后一趟是“1×块数”，合并起来就是 块数$\times(2\times$趟数 $+1)$。

计算“归并趟数”

归并趟数 = $\lceil\log_{M-1}($归并段数$)\rceil$

- M：缓冲区数量（15），$M-1$ 是每趟能合并的归并段数（因为要留1个缓冲区存结果）。
- 归并段数 = $\lceil$块数$\div M\rceil$（把关系拆成多少个“能放进内存的小片段”）

对R排序的代价

1. 计算R的归并段数：$\lceil 1000\div 15\rceil = 67$
   （15×66=990，剩下10块也要拆成1段，所以是67段）

2. 计算R的归并趟数：$\lceil \log_{14}(67) \rceil = 2$（因为 $14^1 = 14 < 67$，$14^2 = 196 \geq 67$，所以需要2趟归并）
3. R的排序代价：
   $1000 \times (2 \times 2 + 1) = 1000 \times 5 = 5000$ 块传输

对S排序的代价

1. 计算S的归并段数：$\lceil 200 \div 15 \rceil = 14$（15×13=195，剩下5块拆成1段，共14段）
2. 计算S的归并趟数：$\lceil \log_{14}(14) \rceil = 1$（$14^1 = 14$，刚好1趟归并）
3. S的排序代价：$200 \times (2 \times 1 + 1) = 200 \times 3 = 600$ 块传输

步骤3：理解“归并连接阶段的代价”（合并两个有序关系）

两个关系都排好序后，连接的过程类似“合并两个有序数组”：

- 同时读R和S的块，按连接属性匹配元组，**每个块只需要读一次**（因为已经有序，不会重复读）。

所以归并连接的代价 = R的块数 + S的块数 = $1000 + 200 = 1200$ 块传输

步骤4：总代价（排序+连接）

总代价 = R的排序代价 + S的排序代价 + 归并连接代价
= $5000 + 600 + 1200 = 6800$ 块传输

总结（一句话说清）

排序-归并连接的代价，就是“把两个关系分别排序的磁盘读写次数”，加上“把排好序的关系合并的磁盘读写次数”，这里算出来是6800次块传输。

要不要我帮你画一个**外部排序+归并连接的流程示意图（文字版）**？

向上取整（200 / 15）

12.5.4.2 代价分析

一旦关系已排序，在连接属性上有相同值的元组是连续存放的。所以已排序的每一元组只须读一次，因而每一块也只须读一次。由于两个文件都只须读一遍（假设所有集合 S_i 均可装入内存），因此可知归并连接算法是高效的。所需磁盘块传输次数是两个文件块数之和：$b_r + b_s$。

假设为每个关系分配 b_b 个缓冲块，那么所需磁盘搜索次数为 $\lceil b_r / b_b \rceil + \lceil b_s / b_b \rceil$。由于磁盘搜索代价远比数据传输高，假设还有额外的内存，因此为每个关系分配多个缓冲块是有意义的。例如，假设对于每个 4KB 的块，$t_T = 0.1$ 毫秒，$t_S = 4$ 毫秒，缓冲区大小为 400 块（或者 1.6 MB），则每 4 毫秒的磁 555 盘搜索时间对应每 40 毫秒的传输时间。换句话说，磁盘搜索时间将只占传输时间的 10%。

针对没有排序好的，还需要额外加上排序的损失：

12.4.1 外部排序归并算法

对不能全部放在内存中的关系的排序称为**外排序**（external sorting）。外排序中最常用的技术是**外部排序归并**（external sort-merge）算法。下面讲述该算法。令 M 表示内存缓冲区中可以用于排序的块数，即内存的缓冲区能容纳的磁盘块数。

12.4.2 外部排序归并的代价分析

下面我们计算外部归并排序的磁盘存取代价。令 b_r 代表包含关系 r 中记录的磁盘块数。在第一阶 548 段要读入关系的每一数据块并写出，共需 $2b_r$ 次磁盘块传输。初始归并段数为 $\lceil b_r/M \rceil$。由于每一趟归并会使归并段数目减少为原来的 $1/(M-1)$，因此总共所需归并趟数为 $\lceil \log_{M-1}(b_r/M) \rceil$。对于每一趟归并，关系的每一数据块读写各一次，其中有两趟例外。首先，最后一趟可以只产生排序结果而不写入磁盘。其次，可能存在在某一趟中既没有读入又没有写出的归并段——例如，某一趟有 M 个归并段需归并，其中 $M-1$ 个被读入并归并，而另一个归并段在该趟归并中却未被访问。忽略后一种特殊情况（相对少地）所能节省的磁盘存取，关系外排序的磁盘块传输的总数为：

$$b_r(2\lceil \log_{M-1}(b_r/M) \rceil + 1)$$

把该公式用到图 12-4 所示的例子上，我们算出共需 $12 * (4+1) = 60$ 次块传输，这一结果可以在图 12-4 中得以验证。注意，上面的值不包括将最后结果写到外存的开销。

此外，我们还要加上磁盘搜索的代价。在产生归并段阶段需要为读取每个归并段的数据作磁盘搜索，也要为写回归并段作磁盘搜索。在归并阶段，如果每次从一个归并段读取 b_b 块数据（也就是说，把 b_b 个缓冲块分配给每个归并段），则每一趟归并需要作 $\lceil b_r / b_b \rceil$ 次磁盘搜索以读取数据[⊖]。尽管输出结果是顺序写回磁盘的，如果它和输入归并段在一个磁盘块上，磁盘头在写回连续的块的间隔中可能已经移到别处。这样我们需要为每趟归并加上总共 $2\lceil b_r / b_b \rceil$ 次磁盘搜索，除了最后一趟以外（因为我们假定最终结果不写回磁盘）。假设输出阶段也分配了 b_b 个块，每一趟可以归并 $\lfloor M/b_b \rfloor - 1$ 个归并段，则磁盘搜索的总次数为：

$$2\lceil b_r/M \rceil + \lceil b_r / b_b \rceil (2\lceil \log_{\lfloor M/b_b \rfloor - 1}(b_r/M) \rceil - 1)$$

如果我们把分配给每个归并段的缓冲块数 b_b 设为 1，把该公式用到图 12-4 所示的例子上，我们算出共需 $8 + 12 * (2 * 2 - 1) = 44$ 次磁盘搜索。

概念简答题

5. 概念简答题

英文问题：Describe the process of cost-based optimization.
中文问题：描述基于代价的查询优化过程。

答案（中英文+解释）

基于代价的查询优化（Cost-Based Query Optimization）是数据库选择“最优查询执行计划”的核心流程，目标是通过计算不同执行方案的代价，选择效率最高的方案。其过程分为3个核心步骤：

步骤1：生成等价的逻辑表达式（Generate logically equivalent expressions）

- **英文**：Use equivalence rules to transform the original relational algebra expression into other logically equivalent expressions.
- **中文**：利用“等价规则”将原始关系代数表达式，转换为其他**逻辑等价**的表达式。
- **解释**：
  等价规则是指“不改变查询结果”的表达式变换规则（比如“选择操作提前”“连接顺序交换”）。例如，`σ_A=5（R × S）`等价于`σ_A=5（R）× S`，通过这类变换可以得到多个“结果相同但执行方式不同”的逻辑表达式。

步骤2：标注表达式以生成备选查询计划（Annotate expressions to get alternative query plans）

- **英文**：For each logically equivalent expression, add implementation details (e.g., which join algorithm to use, which index to scan) to form a concrete query plan.
- **中文**：对每个等价的逻辑表达式，添加**实现细节标注**（比如用哪种连接算法、用哪个索引扫描），形成具体的查询执行计划。
- **解释**：
  逻辑表达式只描述“要做什么”（比如“连接R和S”），标注后明确“怎么做”（比如“用哈希连接算法连接R和S，R用索引扫描，S用全表扫描”）。同一个逻辑表达式可以对应多个不同的物理执行计划。

步骤3：基于预估代价选择最便宜的计划（Choose the cheapest plan based on estimated cost）

- **英文**：Estimate the cost of each alternative plan (using statistics like number of tuples, cost formulae of algorithms), then select the plan with the lowest estimated cost.
- **中文**：基于数据库统计信息（比如元组数量、属性不同值数量）和算法的代价公式，预估每个备选计划的代价，最终选择代价最低的计划。

- **解释**：
  代价通常以“磁盘I/O次数”“CPU时间”等指标衡量（磁盘I/O是主要瓶颈）。例如，通过统计信息预估连接操作的中间结果大小，再用排序-归并连接的代价公式计算其I/O次数，对比不同计划的代价后选最优。

补充：代价预估的依据

代价预估依赖3类信息：

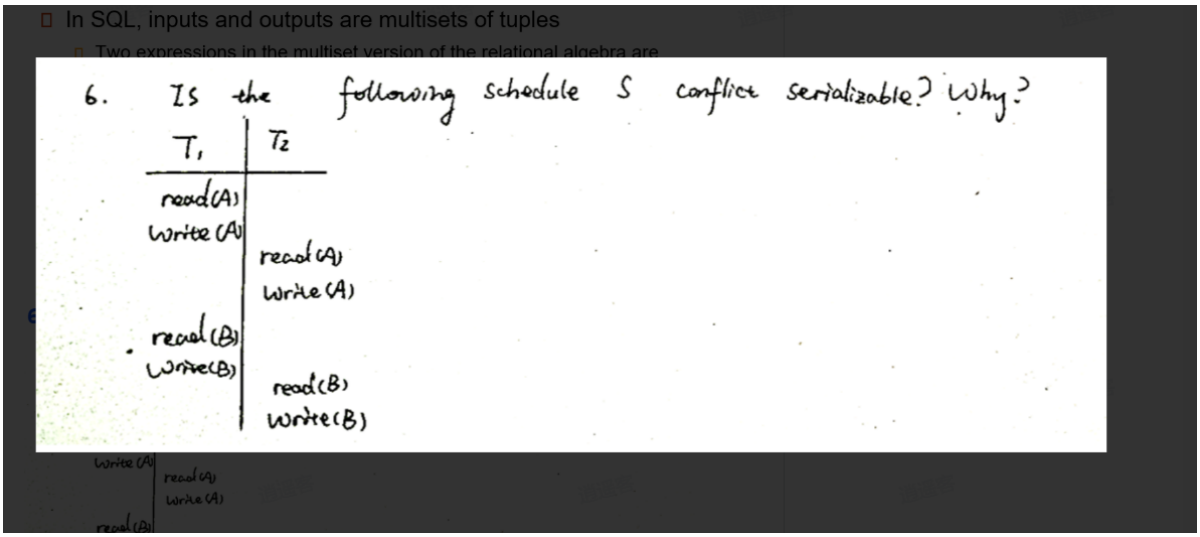
1. 关系的统计信息（如元组数量、块数）；
2. 中间结果的统计信息（如选择操作后的元组数量）；
3. 算法的代价公式（如排序、连接的I/O代价公式）。

要不要我帮你整理一份**基于代价的查询优化步骤的简化流程图（文字版）**？

先生成等价的逻辑表达式、然后对逻辑表达式进行标注生成备选查询计划、接着是根据预估代价选择最便宜的计划！

6. 可串行化调度问题

英文问题：Is the following schedule S conflict serializable? Why?
中文问题：以下调度 S 是冲突可串行化的吗？为什么？



答案与解释

要判断调度是否**冲突可串行化**，核心方法是构造**优先图（Precedence Graph）**，并检查图中是否存在**环**：

- 优先图的顶点：参与调度的事务（本题中是 T_1、T_2）。
- 优先图的边：若事务 T_i 与 T_j 存在**冲突操作**（对同一数据项，且至少一个是写操作），且 T_i 的操作在 T_j 之前执行，则添加一条边 $T_i \to T_j$。

步骤1：分析调度中的冲突操作

本题调度的操作序列（按时间顺序）：

1. T_1 : $\mathrm{read}(A)$
2. T_1 : $\mathrm{write}(A)$
3. T_2 : $\mathrm{read}(A)$
4. T_2 : $\mathrm{write}(A)$
5. T_1 : $\mathrm{read}(B)$
6. T_1 : $\mathrm{write}(B)$
7. T_2 : $\mathrm{read}(B)$
8. T_2 : $\mathrm{write}(B)$

找出**冲突操作对**：

- 对数据项 A：T_1 的 $\mathrm{write}(A)$ 先于 T_2 的 $\mathrm{read}(A)$（冲突，添加边 $T_1 \to T_2$）。
- 对数据项 B：T_1 的 $\mathrm{write}(B)$ 先于 T_2 的 $\mathrm{read}(B)$（冲突，添加边 $T_1 \to T_2$）。

步骤2：构造优先图

优先图的顶点是 T_1、T_2，边是 $T_1 \to T_2$（无反向边）。

步骤3：检查是否有环

优先图中**无环**，因此调度是冲突可串行化的。

最终结论

- **英文**：The schedule S is conflict serializable. Because the precedence graph of the schedule (with vertices T_1, T_2 and an edge $T_1 \rightarrow T_2$) has no cycles.
- **中文**：调度 S 是冲突可串行化的。因为该调度的优先图（顶点为 T_1、T_2，边为 $T_1 \rightarrow T_2$）中不存在环。

要不要我帮你整理一份**冲突可串行化判定的步骤模板**？