数据库试题

目录

1. 九八年秋季试题

1.1.概念题

1.1.1.比较半联接方法和枚举法的优缺点。

半联接技术缩减了连接操作的操作数，以降低通信费用。枚举法适用于缩减局部代价的情况。

评估查询操作的代价需要综合考虑局部代价和传输代价。侧重哪一个方面，需根据系统组成环境确定。如侧重传输代价，局部代价可以忽略不计时，采用半联接技术较好；相反，如果侧重局部代价时，采用直接连接比采用半连接技术优越。因为直接连接技术实现简单，枚举法是基于直接连接的实现方法，此时应采用枚举法。

半连接优点：传输代价低。

半连接缺点：没有考虑局部代价；当“选择度”交低时，半连接技术才可行。

1.1.2. 2PL 协议的基本思想。

2PL协议的基本思想。

并发控制是分布式事务管理的基本任务之一。其目的是保证分布式数据库系统中多个事务的高效正确的执行。有两种模型来实现：其中之一是以“锁”方式为基础的形式模型，一种是以时间印方式为基础的时间模型。

锁方式的基本思想是：事务对任何数据的操作必须先申请数据项的锁，只有申请到了锁之后，即加锁成功以后，才可以对数据项进行操作。操作完成了以后，要释放已经申请的锁。通过锁的共享和排斥的特性，实现事务的可串行化调度。

“锁”又可分为“读锁”和“写锁”：

“读锁”是对数据项进行读操作时要加的锁。由于读操作是可共享操作，所以“读锁”也称为共享锁。

“写锁”是对数据进行写入操作时要加入的锁。写操作是不可共享的锁，因此也叫“排它锁”。

2PL(两阶段锁)协议是并发控制算法中的重要算法。其主要内容是并发执行的多个事务，在对数据进行操作以前要进行加锁，并且每个事务中的所有加锁操作都得在第一个解锁操作以前执行。因此，每个事务中的加锁操作和解锁操作分布在两个部分中，所以此协议称为2PL协议。

在分布式数据库系统中，如果全部的分布式事务均以2PL协议加锁，则系统中各个场地上的局部调度是可串行化的。因为对每个局部场地而言，其上执行的操作只是全局操作的一部分，而全局操作采用2PL协议加锁，显然局部操作也遵循2PL协议。

1.1.3. WAL 协议的主要思想。

系统的故障恢复是以日志文件为基础完成的，因此，要求事务在执行过程中满足先写日志协议（WAL）(**Write_ahead logging**)。当系统发生故障时，可有效地采用重做（redo）和反做（undo）两个基本恢复操作进行恢复。

先写日志协议（WAL）含义：

（1） 在外存数据库被更新之前，应将日志文件中的反做信息写入外存文件；

（2） 事务提交之前，日志文件中的有关重做信息应在外存数据库更新之前写入外存文件。

1.1.4.SSPARC三级模式体系结构。

ANSI/SPARC 提出将数据库系统分为三种层次

一、从数据库管理系统的角度看，可分为三层，从外到内依次为外模式、模式和内模式。

1、 外模式：

外模式又称为用户模式：是数据库用户和数据库系统的接口，是数据库用户的数据视图（View），是数据库用户可以看见和使用的局部数据的逻辑结构和特征的描述，是与某一应用有关的数据的逻辑表示。

2、 模式：

模式是所有数据库用户的公共数据视图，是数据库中全部数据的逻辑结构和特征的描述。

3、 内模式：

内模式又称为存储模式（Storage Schema），是数据库物理结构和存储方式的描述，是数据在数据库内部的表示方式。

二、为了实现三个抽象级别的联系和转换，数据库管理系统在三层结构之间提供了“两层映像”：外模式/模式映像和模式/内模式映像。

1、外模式/模式映像

通过外模式与模式之间的映像把描述局部逻辑结构的外模式与描述全局逻辑结构的模式联系起来。

2、模式/内模式映像

通过模式与内模式之间的映像把描述全局逻辑结构的模式与描述物理结构的内模式联系起来。

1.1.5.设计OID的数据结构时应考虑哪些问题。

OID 对象标识

对象标识在编程语言中已经存在很长时间了，在数据库概念中还是刚刚被提及（1989）。他的概念如下：在一个拥有对象标识的模型中，一个对象的存在并不依赖于他的值。对象标识不应该依赖于它所代表的对象的值。而“对象相等”的概念反映了对象标识与对象的值之间的关系。有两种对象相等的意见存在，一种是两个对象是同一个，即是一个对象那么它的对象标识就是同一个对象标识。还有一种是指两个对象相等，指他们的值相等而对象标识不等。这有两个含义，一个是对象共享，还有一个是对象更新。

对象共享：在一个基于标识的模型中，两个对象可以共享一个部件。这样，复杂对象的图形化表示是一个图，而在一个没有标识的模型中，限制成一个树。考虑下面的例子：一个人有姓名、一个年龄、一个孩子的集合。假设Peter和Susan都有一个15岁的孩子，叫john。在实际生活中，会有两种情况，一是Peter和Susan是同一个孩子的父母，或者是有二个孩子，在没有标识的模型中，Peter表示成：

(Peter, 40, {(John, 15, {})})

Susan表示成

(Susan, 41, {(John, 15, {})})

这样，没有方法表示是否Peter和Susan是同一个孩子的父母。在一个基于标识的模型，那两个结构是否共享同一个部分(John, 15, {})。

对象更新：假设Peter和Susan确实是John的父母，在这种情况下，所有对于Susan孩子的更新操作也会影响到Peter的孩子。在一个基于值的系统，这两个子对象必须都要更新。对象标识同时也是强大的数据操纵的基本要素，可以时集合、元组的基础和递归的复杂对象操纵。

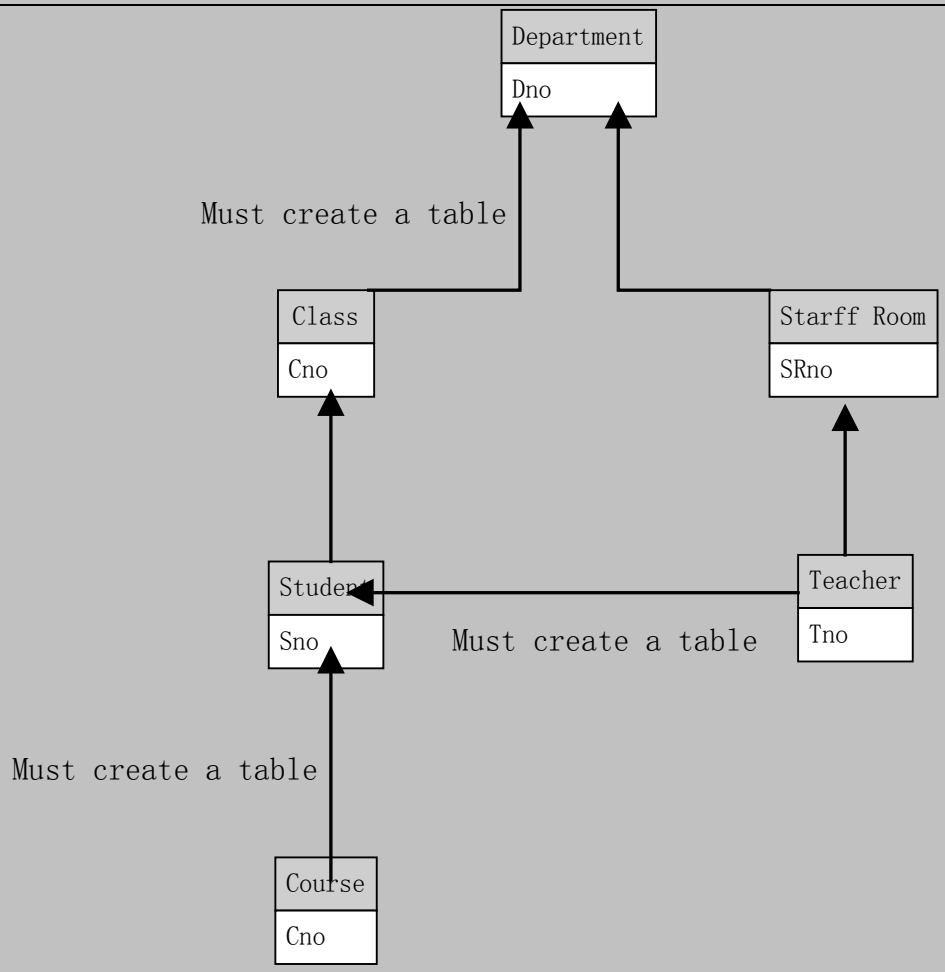
支持对象标识意味着提供对象赋值功能、对象拷贝（深拷贝和浅拷贝）、对象标识检测和对象比较等（深相等和浅相等）。

当然，用户可以在基于值的系统上，显式给定对象标识来模拟。然而，这种方式会给用户确保对象标识的唯一性和参照完整性上增加负担，特别是类似于垃圾回收的操作。

要注意的是对象标识在编程语言中是必须的，程序中操纵的每个对象都有一个标识可以被更新。这个标识有时是一个变量名，有时也是物理内存中的一个地址。但是在纯关系型模型中，是没有这一概念的，因为它是一个基于值的模型。

1.2.某个大学中有若干系，且每个系有若干个班级和教研室，每个教研室有若干个教员，其中教授、副教授，每个人带领若干名研究生。每个班有若干名学生，每个学生可选修若干门课程，每门课程可由若干学生选修。完成下列各种要求：

(1) 设计一个概念模式以描述上述情况，画出E-R图。要求补充每个实体的适当属性



(2)将“E-R图”转换为关系模式

答：系（系号,系名）= Dept(Dno,Dname)

班级（班级号,班级名）= Class(Cno,Cname)

教研室（教研室号,教研室名,系号）= StaffRoom(SRno,SRname,Dno)

学生（学生号,学生名,班级号）= Student(Sno,Sname,Cno)

课程（课程号,课程名）= Course(Cono,Coname)

教员（教员号,教员名,职位,工资号,系号）=Teacher(Tno,Tname,Position,WageNo, Dname)

选课（课程号,学号）= SelectCouse(Cono,Sno) //多对多的关系必须建立一个表。

班系（班级号,系号）= ClassDept(Cono,Dno)//一对多的关系可建立也可不建立看情况。

选导师（学号,教员号）=SelectTeacher(Sno,Tno) //一对多的关系可建立也可不建立看情况。在本题需要建立。

(3)使用SQL语言来完成下列查询

(a) 查找计算机系的，“学号”为981001的学生选修的课程名称和课程号。

```
SELECT Cono,Coname
FROM   Course
WHERE   EXIST
```

(SELECT *
FROM SelectCouse
WHERE Sno = “981001”
)

(b) 求没有带研究生的所有教授或副教授的名称、工资号及所在的系名。

SELECT Tname, WageNo,Dname
FROM Teacher
WHERE Teacher. Position:professor or vice-professor AND NOT EXIST
(SELECT *
FROM SelectTeacher
)

(c) 至少有一个班级的系的名称。

SELECT Dname
FROM Dept
WHERE EXSIT
(SELECT *
FROM ClassDept
)

1.3. 下面是某学院的一个学生档案数据库的全局模式：

Student={sno,sname,sex,birth,major,class,dno}
Grade={sno,cno,grade}

其中sno,sname,sex,birth为学生的“学号”，姓名，性别和出生日期，major,class,dno为专业，班级和所属的“系号”。全学院共有8个系，dno分别为1，2，…8。其中cno为课程号，grade为考试成绩。根据需要，要求将学生关系Student中的{sno,class }保存在场地0，其他属性和成绩Grade按“系号”1－4，5－6分别保存在场地1和场地2上。根据上述要求，

1.3.1.将全局模式进行分片，写出分片定义和分片条件。

分片定义和分片条件

Stu = Student ∞ Grade
sno=sno

Student1= $\Pi_{sno,class}$ Stu

Temp= $\Pi_{sname,sex,birth,major,dno}$ Stu

Student2= $\delta_{1<=dno<=4}$ Temp

Student3= $\delta_{5<=dno<=6}$ Temp

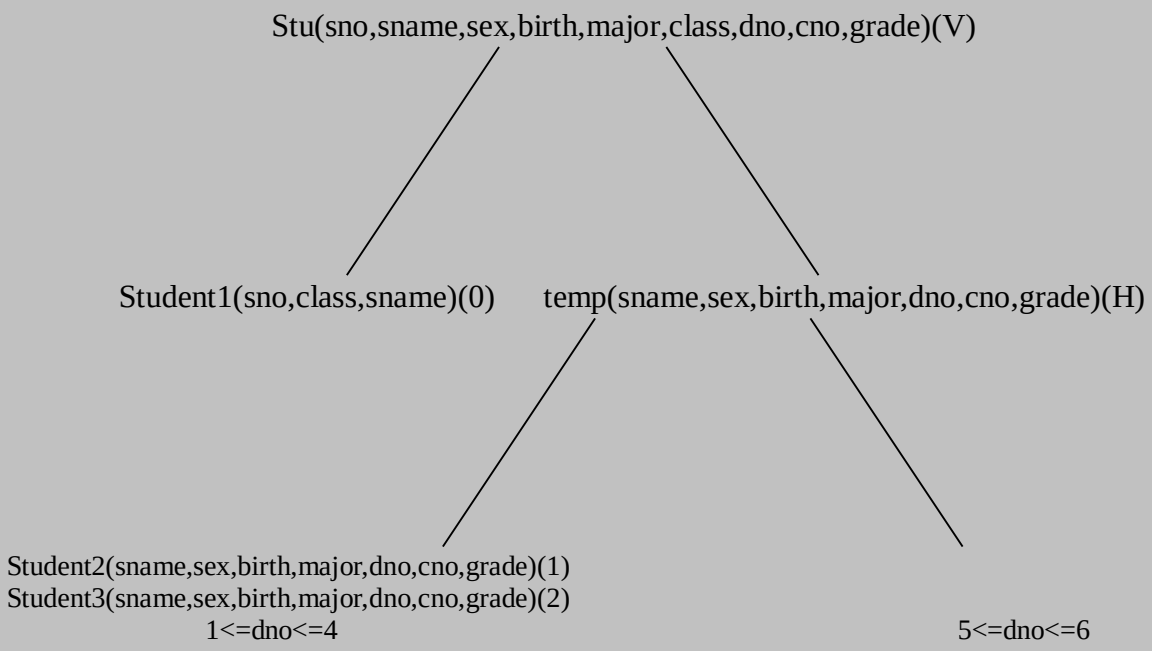
Site Student1(0), Student2(1), Student3(2);

1.3.2.指出各分片的类型，并画出“分片树”。

Student1= $\Pi_{sno,class,sname}$ Student是垂直分片

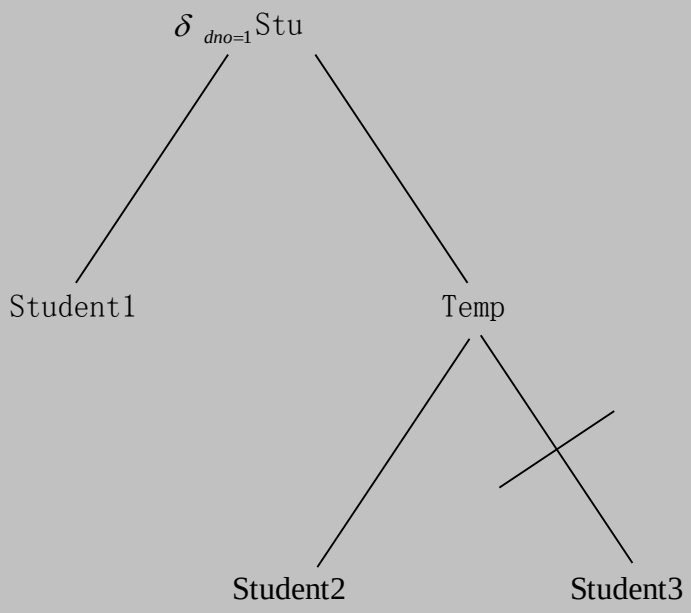
Student2= $\delta_{1<=dno<=4}$ Temp是水平分片

Student3=$\delta_{5<=dno<=6}$ Temp是水平分片

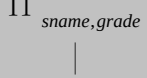


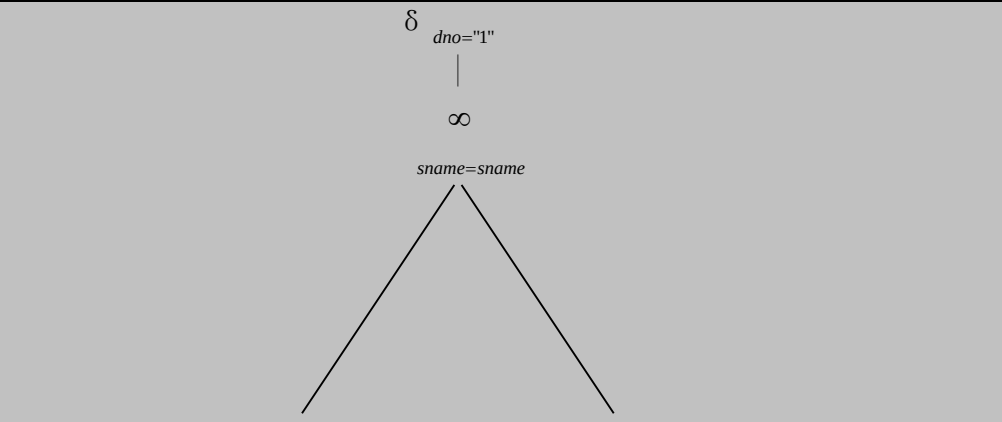
1.3.3.假设要求查询“系号”是 1 的所有学生的姓名和成绩，写出在全局模式上的SQL查询语句，并要求转换成相应的关系代数表示，画出全局查询树，请依次进行全局优化和分片优化，画出优化后的查询树。要求给出优化变换过程。

```
SELECT   *
FROM     Stu
WHERE     dno = “1”
```

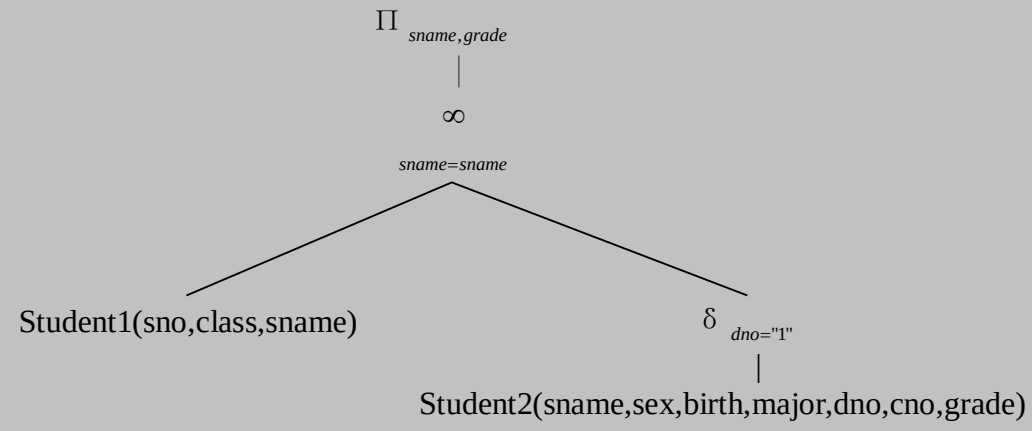


去掉Student3如图





按C1、C2准则转换将一元操作下移。



1.4. 设数据项x,y存放在S1 场地，u,v存放在S2 场地，有分布式事务T1 和T2,T1 在S1 场地的操作为R1(x)W1(x)R1(y)W1(y),T2 在S1 场地的操作为R2(x)R2(y)W2(y);T1 在 S2 场地上的操作作为R1(u)R1(v)W1(u),T2 在S2 场地上的操作作为W2(u)R2(v)W2(v)。对下述 2 种情况，各举出一种可能的局部历程（H1 和H2），并说明理由。

串行调度	数据项x,y		数据项u,v	
Time	Site S1		Site S2	
	T1	T2	T1	T2
	R1(x)		R1(u)	
	W1(x)		R1(v)	
	R1(y)		W1(u)	
	W1(y)			
		R2(x)		W2(u)
		R2(y)		R2(v)
↓		W2(y)		W2(v)

T1 < T2

1.4.1.局部分别是可串行化，而全局是不可串行化的

数据项x,y	数据项u,v

Time	Site S1		Site S2	
|	T1	T2	T1	T2
|	R1(x)			W2(u)
|	W1(x)		R1(u)	R2(v)
|	R1(y)	R2(x)		W2(v)
|	W1(y)		R1(v)	
|		R2(y)	W1(u)	
|		W2(y)		
↓				

$T1^{S1} < T2^{S1}$ $T2^{S2} < T1^{S2}$

T1、T2的所有子事务在每个站点都是可串行执行的。但

根据2PL协议事务T1在没有获得对v的锁之前是不会释放y的锁，而T2在没有获得y的锁之前是不会释放v的锁，T1和T2发生了死锁，故T1和T2之间在全局上是不可串行化的。

1.4.2.局部和全局都是可串行化的。要求按照严格的 2PL协议，加上适当的加锁和解锁命令，（注意，用rl(x)表示加读锁，wl(x)表示加对x加写锁，ul(x)表示解锁）

全局事务在全局范围内是可串行化的，必须是全局事务的所有子事务在每个局部站点上的可串行性在调度表中出现的顺序必须相同。即Ti^A<Tj^A,则对于所有拥有Ti和Tj代理的站点k都有Ti^k<Tj^k,

Time	Site S1 数据项x,y		Site S2 数据项u,v	
|	T1	T2	T1	T2
|	rl1(x)		rl1(u)	
|	R1(x)	RL2(x)	R1(u)	wl2(u)
|	wl1(x)	wait	rl1(v)	wait
|	rl1(y)	.	R1(v)	.
|	R1(y)	.	wl1(u)	.
|	W1(y)		w1(u)	.
|	U1(x)	.	u1(u)	
|	U1(y)	.	u1(v)	
|		rl2(x)	wl2(u)	
|		R2(x)	w2(u)	
|		rl2(y)	rl2(v)	
|		R2(y)	r2(v)	
|		wl2(y)	wl2(v)	
|		W2(y)	w2(v)	
|		U2(x)	u2(u)	
|		U2(y)	u2(v)	
↓				

1.5. 试述面向对象的数据库系统中页面服务器和对象服务器两种Client/Server体系结构的主要特点,

基于页面服务的体系结构可分为两种：

一种是基于 page to object 的服务体系结构，这种体系结构的特点是数据传输单位是页面，在客户方缓冲页面和对象，其优点是服务器的性能大大的提高了，而且在客户和服务器之间数据传输较少，其缺点方法不能在服务器上运行；另一种体系结构是基于

page to page 服务器体系结构，这种体系结构与前面的体系结构非常相似，差别是客户方仅缓冲页面但不缓冲对象。

基于对象服务器(OS object server)的OODBMS体系结构的特点是客户器与服务器间的数据传输单位是对象，其优点是方法在对象服务器上运行且客户方和服务器方均可进行查询处理，但这种体系结构的缺点是数据传输量可能比较大，而服务器仍然是一个系统性能的瓶颈。

2. 九九年春季试题

2.1.DBMS解决了信息处理技术中的哪些挑战？

关系数据库的基本理论已经成熟,但各大公司在关系数据库管理系统(RDBMS)的实现和产品开发中，都遇到了一系列技术问题，主要是在数据库的规模愈来愈大，数据库的结构愈来愈复杂，又有愈来愈多的用户共享数据库的情况下，如何保障数据的完整性(Integrity)、安全性(Security)、并行性(Concurrency)，以及一旦出现故障后，数据库如何实现从故障中恢复(Recovery)。这些问题如果不能圆满解决，无论哪个公司的数据库产品都无法进入实用，最终不能被用户所接受。正是在解决这些重大的技术问题，促使了 DBMS 发展和成熟。

目前，DBMS 解决上述问题的主要技术手段和方法如下：把对数据库的操作划分为称作"事务"(transaction)的原子单位，对 1 个事务内的操作，实行"all or not"的方针，即"要么全做，要么全不做"。用户在对数据库发出操作请求时，系统对有关的不同程度的数据元素(字段、记录或文件)"加锁"(locking)；操作完成后再"解锁"(unlocking)。对数据库的任何更新分两阶段提交(2PL)。建立系统运行日志(log)，以便在出错时与数据库的备份(backup)一起将数据库恢复到出错前的正常状态。

上述及其他各种方法可总称为"事务处理技术"(transaction processing technique)。事务处理技术虽然诞生于数据库研究，但对于分布式系统，client/server 结构中的数据管理与通信，对于容错和高可靠性系统，同样具有重要的意义。

2.2.在关系数据库应用设计中，为什么要对数据库模式进行规范化？

关系数据库的设计是对数据进行组织化和结构化的过程，核心问题是关系模型的设计。关系模型是数学化的、用二维表格数据描述各实体之间的联系的这么一种模型；它是所有的关系模式、属性名和关键字的汇集，是关系模式描述的对象。简单的说关系模式是指一个关系的属性名表，即“二维表”的“表框架”。关系模式的设计是关系模型设计的灵魂。所以，关系模式的设计是关系数据库设计核心的核心。

关系模式的设计直接决定着关系数据库的性能。目前，在指导关系模式的设计中规范化(normalization)设计占有主导地位，它是在数据库几十年的长期发展中产生并成熟的。但近年来这一领域出现了一种新的趋势，一种称为“非规范化”(denormalization)的关系模式设计引起业界的关注并已在一定的范围内得到应用。对这一新的设计思想，各方反应迥然不同褒贬不一，从而在相关的理论界掀起了一场不大不小的规范化与非规范化之争。本文简单介绍了规范化与非规范化设计的基本思想，综述了正反双方争论的要点，供参考。

一、规范化设计

关系模式规范化设计的基本思想是通过对关系模式进行分解，用一组等价的关系子模式来代替原有的关系模式，消除数据依赖(包括函数依赖和多值依赖)中不合理的部分，使得一个关系仅描述一个实体或者实体间的一种联系。这一过程必须在保证无损连接性、保持函数依赖性的前提下进行，即确保不破坏原有数据的前提下，并可将分解后的

关系通过自然联接恢复到原有的关系。

具体地说，规范化设计的过程就是按不同的范式，将一个二维表不断地分解成多个二维表并建立表之间的关联，最终达到一个表唯一描述一个实体或者实体间的一种联系的目标。目前遵循的主要范式包括1NF、2NF、3NF、BCNF、4NF和5NF等几种；在工程中3NF、BCNF应用得最广泛，推荐采用3NF作为标准。

规范化设计的优点包括可有效地消除数据冗余，理顺数据的从属关系，保持数据库的完整性，增强数据库的稳定性、伸缩性、适应性。通常认为规范化设计存在的主要问题是增加了查询时的连接“库表”的运算，导致计算机时间、空间、系统及运行效率的损失。在大多数情况下，这一问题可通过良好的索引设计等方法得到解决。

二、非规范化设计

非规范化设计的基本思想是，现实世界并不总是依从于某一完美的数学化的关系模式。强制性地对事物进行规范化设计，形式上显得简单化，内容上趋于复杂化，更重要的是导致数据库运行效率的减低。非规范化要求适当地降低甚至抛弃关系模式的范式，不再要求一个表唯一描述一个实体或者实体间的一种联系。其主要目的在于提高数据库的运行效率。

非规范化处理的主要技术包括增加冗余或派生列，对表进行合并、分割或增加重复表。一般认为，在下列情况下可以考虑进行非规范化处理：(1)大量频繁的查询过程所涉及的表,都需要进行连接的时候；(2) 主要的应用程序在执行时要将表连接起来进行查询的情况下；(3)对数据的计算需要临时表或进行复杂的查询时。

非规范化设计的主要优点是减少了查询操作所需的连接；减少了外部键和索引的数量；可以预先进行统计计算，提高了查询时的响应速度。非规范化存在的主要问题是增加了数据冗余；影响数据库的完整性；降低了数据更新的速度；增加了存储表所占用的物理空间。其中最重要的是数据库的完整性问题。这一问题一般可通过建立触发器、应用事务逻辑、在适当的时间间隔运行批命令或存储过程等方法得到解决。

三、规范化与非规范化争论的要点

支持非规范化设计的一方认为，数据库规范化的程度越高，其中表的数量越多，规范化的程度与表的数量直接相关；表的数量越多，表的连接运算也越多；连接运算增多，必然降低数据库执行的速度，影响数据库的性能。只有通过非规范化设计，显著减少表的数量，从而减少对连接运算的依赖，加速数据库执行的速度，才能保证数据库性能的正常发挥。例如目前流行于决策支持系统的非规范化星型模式就远胜于应用规范化设计，是非规范化设计的最好范例。非规范化设计并不意味着混乱和无视规则，它也遵循保护信息完整性等软件工程的基本原则。

支持规范化设计的一方认为，规范化与非规范化只是一个逻辑概念，强调非规范化设计者混淆了逻辑与物理的关系。数据库的性能是由物理水平决定的，即硬件、数据库的大小和物理设计、数据存储和访问的方法、数据库管理系统的优化程度、并发访问的数量等；非规范化设计并未改变数据库的物理水平，因此不可能提高数据库的性能。规范化并不只是为了避免数据冗余，更重要的是为了确保数据库的完整性。非规范化设计的最大问题是难以保证数据库中数据的一致性，存在着破坏数据的危险。此外，非规范化使一个表中存在多个实体，不同实体混合在一起强化了数据库的复杂性，提高了用户理解的难度，并导致描述问题上的困难，增加了正确响应的风险。只有规范化设计才是解决这些问题的根本途径。如果不摒弃非规范化设计理念，为了获得所谓的性能的提高而漠视数据库完整性被破坏的风险，就无法激励“开发商”去研究真正的完全规范化而高性能的关系数据库管理系统，其后果必然影响数据库的健康发展。

从某种意义上说，数据库的规范化与非规范化设计并不是对立的、非此即彼的关系。也许其中一方会逐渐消亡，也许二者存在一条中间道路可走。认识事物原本存在一个螺

旋式上升的过程。这场争论尚未结束，也无法对最终的结果进行预测。但可以肯定的是，无论结果如何，都将对未来数据库的发展方向产生深远的影响。

2.3.简述ACID特性。

事务具有四个特性：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）和持续性（Durability）。这个四个特性也简称为ACID特性。

原子性（Atomicity）:事务是数据库的逻辑工作单位，事务中包括的诸操作要么都做，要么都不做

一致性（Consistency）：事务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态。因此，当数据库只包含事务成功提交的结果时，就说数据库处于一致性状态之中。如果数据库系统运行中发生故障，有些事务尚未完成就被迫中断了，系统就将此事务中对数据库的所有已经完成的操作全部撤消，滚回到事务开始时的一致状态。

隔离性（Isolation）：一个事务的执行不能被其他事务干扰，即一个事务的内部操作及其使用的数据对于其他并发事务来说是隔离的。并发执行的各个事务之间不能互相干扰。

持续性（Durability）：持续性也称永久性（Permanence），指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其执行结果有任何影响。

事务是恢复和并发控制的基本单位。保证事务ACID特性是事务处理的重要任务。事务ACID特性可能遭到破坏的因素有：

1. 多个事务并行运行时，不同事务的操作交叉执行。
2. 事务在运行过程中被强行停止。

在第一种情况下，数据库管理系统必须保证多个事务的交叉运行不影响这些事务的原子性。在第二种情况下，数据库管理系统必须保证被强行终止的事务对数据库和其它事务没有任何影响。

这些就是数据库管理系统中恢复机制和并发控制机制的责任。

2.4.长事务处理有哪些特性，如何解决？

通常DBMS中采用"锁定-修改-释放"的策略以实现其对多用户并发操作数据库的控制。但这种策略不很适合用于处理地理数据的DBMS。对地理数据的编辑工作，可以几分钟做完，也可能要拖上几个月。这种情形即是所谓"长事务处理"。ArcSDE 8对长事务处理提供了底层的支持。当ArcSDE服务器的一个实例（instance）第一次启动时，就建立了数据库缺省的状态和版本。用户可在此基础上建立公共的或私有的数据版本。用户各自在自己的数据版本上工作，因而无须对多个用户同时访问的数据对象进行锁定。

每个用户都是在直接对数据库进行操作（编辑、修改），但是ArcSDE为其建立了记录所有修改"痕迹"的增量记录，即版本。用户在这个数据版本进行编辑修改时，并不用关心其他用户是不是也在对同一数据进行操作。只有当用户完成了他的（长）事务处理工作时，系统才将其当时的数据版本"合并"到原来的数据版本中去，"冲突"也是在此时再加以处理。系统为用户提供了解决冲突的三种选择：维持原状、否决自己的修改或否决别人的修改。

长事务(long transaction) 的应急之策

事务(transaction)是一个不可分的工作单位, 一个事务中包含的操作要么全部执行,要么全部不执行. 如果数据库中只包含成功事务提交的结果, 则数据库的状态称为"一致状态". 为了保证数据库"一致状态", Informix-online采取如下机制:

对于带日志的数据库(logged database)，在事务运行过程中，系统将每个更新操作都记录到日志文件中，并且包含事务开始标记、事务结束标记或“事务回滚”标记。如果该事务在运行过程中出现故障，那么恢复子系统将对该事务做撤销处理(undo)。具体做法就是反向阅读日志文件，对该事务的每一个更新操作做反操作。注意，“事务回滚”作为事务一部分也要向逻辑日志写纪录。当一个事务很大时，有可能在其运行过程中横跨所有的日志文件。此时所有的逻辑日志文件都包含该活动事务，包含活动事务的逻辑日志文件是不能被释放的。如果此事务还没有结束,或者因为出错需要“回滚”，那么就需要继续写逻辑日志而此时没有空闲的日志空间可被利用，就会造成了系统瘫痪。Informix管理系统为了避免此类情况发生定义了一个参数LTXHWM,一般情况下，设置它的值为50，即一个事务横跨的逻辑日志达到整个逻辑日志模块的50%就被定义 为长事务.系统将强制其“回滚”(rollback).

由于存储空间的有限性，业务的快速增长,，升级步伐没有跟上，我们银行业务中长事务偶尔会在关键时出现，如年终结转、季度结息、删除大数据表、卸数重新装载数据库等操作。解决这类问题最根本的办法就是增加逻辑日志空间。但有时事出突然，临时不能解决，又不能影响正常营业，必须当时解决问题。权益之计可以取消数据库的带日志状态，改为 no logged. 此状态下事务的更新操作不再往逻辑日志文件中写，给我们带来的风险是：事务运行过程中出现故障时，恢复子系统就不能“回滚”(roll back)该事务，数据库就被置于"不一致状态"。 但只要按照一定的步骤来处理。就能把风险降到最低或者没有风险。

2.5.数据库系统体系结构有哪几类，每种类型的特点是什么，关键技术有哪些？

数据库系统分为：集中式数据库系统、客户 / 服务器数据库系统、并行数据库系统、分布式数据库系统、

集中式数据库系统的特点：将物理数据集中存放在主机上，由主机上的数据库管理系统统一管理整个数据库，用户可从终端上发出数据操作命令，经主机上的数据库管理系统接收处理后，再将所需数据送回终端。集中式数据库系统技术已成熟，数据共享能力、恢复能力较强，但所有数据库的操作均由主机管理，主机负荷较大，一旦主机故障，则导致系统全部瘫痪。

分布式数据库系统使各部门经常要用的数据能够就近存放，从而分散了工作负荷，可靠性较高，局部发生故障不至于引起整个系统瘫痪。分布式数据库系统它在地理位置上是分布在一个含有多个数据库管理系统的计算机网络中，这些数据库管理系统构成分布式数据库管理系统，在分布式数据库管理系统中，每个用户所使用的数据可以不存储在自己使用的计算机上，而是由分布式数据库管理系统在机器之间通过网络从其它机器上传输过来。

客户/服务器数据库系统将数据处理任务分为两个系统，客户机运行数据库应用，数据库服务器运行实际的数据库管理系统(DBMS)。

并行数据库系统是指利用多处理器平台的能力，同时运行多个事务处理，具有支持分布式操作、多线程处理、联机事务处理和决策处理等能力，从而提高数据库系统的响应时间和事务吞吐量。

① 并行数据库的体系结构：并行数据库的体系结构有3种；共享内存、共享磁盘和无共享资源结构。

② 并行处理技术：并行数据库系统通常采用多进程多线索结构，提供以下4种不同粒度的并行性：

- 不同用户事务间的并行性；
- 同一事务内不同查询间的并行性；
- 同一查询内不同操作间的并行性；

• 同一操作内的并行性。

③ 线程和进程：在一个事务处理中，每个用户的连接都能由操作系统进程或线程〔由操作系统生成的工作单元〕分别完成。由于线程使用很少的内存和CPU资源，因此线程的效率比“进程”的效率高。

2.6.决策支持类应用与OLTP应用对于数据库系统的要求有哪些不同，支持前者的关键技术有哪些，并简述之。

OLTP:On-Line Transaction Processing在线事务处理。

DSS Decision Support System 决策支持系统。DSS是在管理信息系统(MIS)基础上发展起来的。MIS是利用数据库技术实现各级管理者的管理业务，在计算机上进行各种事务处理工作。DSS则是要为各级管理者提供辅助决策的能力。

决策支持系统主要是以模型库系统为主体，通过定量分析进行辅助决策。其模型库中的模型已经由“数学模型”扩大到数据处理模型、图形模型等多种形式，可以概括为广义模型。决策支持系统的本质是将多个广义模型有机组合起来，对数据库中的数据进行处理而形成决策问题大模型。决策支持系统的辅助决策能力从运筹学、管理科学的单模型辅助决策发展到多模型综合决策，使辅助决策能力上了一个新台阶。

OLTP 数据库	DSS 数据库
OLTP = online transaction processing	DSS = data warehousing
联机事物处理	数据仓库
例如：飞机订票,网上交易,BBS等	例如：各种资源资料查询系统
大量的在线用户和DML操作	很少的DML操作
大量基于索引的查询	大量的全表扫描的查询
用B-tree，reverse key索引，定期索引重建	用bitmap索引
需要较多的小的“回退段”	需要较少的大的“回退段”
不要用分布式查询	用分布式查询
数据对象的存储参数pctfree 20 或者更高	数据对象的存储参数pctfree 0
共享程序代码和各种变量常量	字符变量和线索
启动多线索服务	使用大的数据块,db_file_mutiblock_read_count
使用较大的日志文件	使用较小的日志文件
listener开多个响应端口	增加sort_area_size

注：(数据操作语言DML,即Data manipulation Language)

联机分析处理 (OLAP) 的概念最早是由关系数据库之父E.F.Codd于1993年提出的，他同时提出了关于OLAP的12条准则。OLAP的提出引起了很大的反响，OLAP作为一类产品同联机事务处理 (OLTP) 明显区分开来

当今的数据处理大致可以分成两大类：联机事务处理OLTP（on-line transaction processing)、联机分析处理OLAP（On-Line Analytical Processing)。OLTP是传统的关系型数据库的主要应用，主要是基本的、日常的事务处理，例如银行交易。OLAP是数据仓库系统的主要应用，支持复杂的分析操作，侧重决策支持，并且提供直观易懂的查

询结果。下表列出了OLTP与OLAP之间的比较。

	OLTP	OLAP
用户	操作人员,低层管理人员	决策人员,高级管理人员
功能	日常操作处理	分析决策
DB 设计	面向应用	面向主题
数据	当前的，最新的细节的，二维的分立的	历史的，聚集的，多维的集成的，统一的
存取	读/写数十条记录	读上百万条记录
工作单位	简单的事务	复杂的查询
用户数	上千个	上百个
DB 大小	100MB-GB	100GB-TB

2.7.面向对象的数据库是如何产生的，其基本原理是什么？有哪些创新特性？

面向对象的数据库

OODB是面向对象方法与数据库技术的结合领域。OODB既需要具备传统数据库的必备功能，也需要具有面向对象方法的基本特征。例如：采用面向对象的数据模型，把对象作为存储与检索单位，支持继承与封装，提供面向对象的数据定义语言和数据操纵语言等等。目前OODB的研究与开发有三种途径：

.在传统的数据库管理系统中增加面向对象的功能；

.在面向对象的编程语言中增加数据库的功能；

.开发全新的、自成体系的面向对象的数据库。OODB是目前的一个研究热点，需要解决的问题既有理论方面的也有技术方面的。

有些人把面向对象的数据库设计(即数据库模式)思想与面向对象数据库管理系统(OODBMS)理论混为一谈。其实前者是数据库用户定义数据库模式的思路,后者是数据库管理程序的思路。用户使用面向对象方法可以定义任何一种DBMS数据库,即网络型、层次型、关系型和面向对象型等都可以,甚至文件系统设计也照样可以遵循面向对象的思路。

面向对象的思路或称规范可以用于系统分析、系统设计、程序设计,也可以用于数据结构设计、数据库设计。OOSE（面向对象软件工程）自上至下、自始至终地贯彻面向对象思路,是一个一气呵成的统一体。面向对象的数据库设计只是OOSE（面向对象软件工程）的一个环节

2.8.$r_i \propto r_j$一定等于$r_j \propto r_i$吗？在什么条件下$r_i \propto r_j = r_j \propto r_i$成立?

2.9.为了设计一个健壮的分布式系统，你必须知道可能发生哪种类型的失败。

2.9.1.请列出在分布式系统中可能的失败类型：

这里所说的失败就是指故障。分布式数据库可能发生的故障有：在局部数据库中发生“数据丢失故障”及通讯系统故障。

通讯系统故障又包括“报文丢失”和“网络分割”。

局部数据库数据丢失主要由存储介质的故障引起的故障，或是主存中的数据丢失或

出错；或是辅助存储器中的数据丢失或出错；或是命令无法执行而数据没错等。

报文丢失是指传送过程中报文的丢失导致数据不正确。这时必然得不到一个肯定的回答，或者报文正确传送了，而应答信息未正确传到。这种报文丢失造成系统的等待状态可以通过一个有限的协议加以解决，即传送报文的数目是有限的。传送的过程一定在有限的时间内完成。因此当一段时间的延迟以后仍收不到回答则认为报文丢失，此时将重发报文。若重新发送若干次后仍然得不到回答，则认为网络发生故障或场地故障。

网络分割指通讯网络中一部分场地和令一部分场地之间完全失去联系，这时称为网络分割。

2.9.2.在你列出的失败类型中，哪些也可能发生在集中式系统中？

局部数据库数据丢失也可发生在集中式的数据库中。

2.9.3.对于每一种失败类型，在失败发生情况下，两段提交机制如何保证事务的原子性？

《分布式数据库》P123场地故障。

3. 九九年秋季试题

3. 1. 问答题

3.1.1.分布式数据库系统在系统结构、模式结构、功能模块等方面有何特点？

分布式数据库系统的系统结构

分布式数据库系统的系统结构是C/S模式。在网络环境中，每个具有多用户处理能力的硬件平台都可以成为服务器，也可成为工作站。多个服务器上的数据库对用户来说是一个逻辑上的单一数据库整体，数据一致性、完整性及安全性都是对这一逻辑上的单个数据库进行控制的。服务器对共享数据的存取进行管理，而非数据库管理系统的处理操作可以由客户机来完成。

分布式数据库系统的模式结构

在集中式数据库系统中为了实现较高的数据逻辑独立性和物理独立性，一般采用三级模式体系结构。在分布式数据库系统中为了实现场地透明（分布透明），具有更多级模式。

分布式数据库系统的模式结构分为两部分：集中式数据库的模式结构，代表各场地上局部数据库系统的模式基本结构；分布式数据库系统增加的模式级别，包括全局外模式、全局概念模式、分片模式及分布模式。

（1）全局外模式：它为全局概念模式的子集，表示全局应用所涉及的数据部分。

（2）全局概念模式：它定义分布式数据库的全局逻辑结构。

（3）分片模式：由于分布式数据库的数据可按集中、重复、分割混合方式分布，分片模式用于说明如何放置数据库的特殊部分。分布式数据库可划分逻辑片，逻辑片可以是数据项分割，也可以是数据值分割，或是上述两种。逻辑片可以任意定义，但它们之间是相关的。它们组成不相交或者是不重迭的数据库子集。分片模式定义片段、片段与概念模式之间的映像。

（4）分布模式：它定义片段存放结点。一个片段在物理上可分布到不同结点。根据分布模式提供的信息，一个全局查询可分为若干子查询，每一个子查询所访问的数据属于同一场地的局部数据库。

分布式数据库系统中所增加的模式和相应的映像,使分布式数据库系统具有分布透

明性。

分布式数据库系统的功能模块

3.1.2.给出两种 2PL协议，并比较它们的优点缺点？

2PL协议适用条件很简单，要求一个事务内部的所有加锁操作应在所有撤消锁操作之前发生。

数据库系统中采用的常规并发控制技术(如2PL)在不适合工作流系统的情况下，如果采用2PL协议，则所涉及的资源将被长期阻塞，而导致整个系统可用性极低。我们采用分层事务模型的方法，提前提交子事务(任务)，释放被封锁住的资源，避免底层子事务(任务)冲突造成的阻塞。这样，任务的正确性由任务调度系统负责，工作流的正确性由工作流系统负责。并且，在定义工作流时，用户必须向系统提供关于工作流的正确性的语义信息。

3.1.3.解释为什么对象类的多继承存在二义性，并通过例子加以说明。

P112,“虚基类”。

3.1.4.对于下述情况，哪种并行性（查询间并行性、操作间并行性、操作内并行性）有助于正加系统的吞吐量：

查询间并行处理：是指在多处理器和多磁盘上并行执行单个查询。这种并行处理对于加快运行时间较长的查询来说是十分有效的。而在每个查询都是串行执行的任务中，查询间并行处理是没有什么效果的。查询间并行处理有利于提高单个查询的响应时间，而对提高系统的吞吐能力并没有多大帮助作用。

单个查询能以如下两种方式执行：

操作间并行处理:并行执行一个查询表达式中的不同操作，从而加速执行一个查询。

操作内并行处理:并行执行查询内的每个独立的操作，如排序、选择、投影和连接，从而加速该查询的处理速度。

据此推断：

(a) 具有大量小查询的时。

据此推断：具有大量小查询时，应该用查询间并行性

(b) 有些查询非常大且系统的磁盘和处理机数量很多时。

有些查询非常大且系统的磁盘和处理机数量很多时应该用操作间并行性和操作内并行性。

3.2. 下面是某个公司人事数据库的两个全局关系

EMP={eno,ename,title,salary,addr,phone,dno};DEPT={dno,dname}该公司共有3个部门，dno分别为0，1，2。要求将DEPT关系和EMP关系的部分属性（ename,addr,phone）保存在部门0的场地上，EMP关系的部分属性（title,salary）保存在所在部门场地上。根据上述要求，

3.2.1.将全局模式进行分片，写出分片定义和分片条件。

答：对关系PAY分片

分片定义和分片条件

EMP1=$\Pi_{ename,addr,phone,dno}$EMP

EMP2=$\Pi_{ena,title,salary,dno}$EMP

Site EMP1(0),DEPT(0);

TEMP0=$\delta_{dno=0}$ EMP2

TEMP1=$\delta_{dno=1}$ EMP2

TEMP2=$\delta_{dno=2}$ EMP2

EMP20=$\Pi_{title,salary}$ TEMP0

EMP21=$\Pi_{title,salary}$ TEMP1

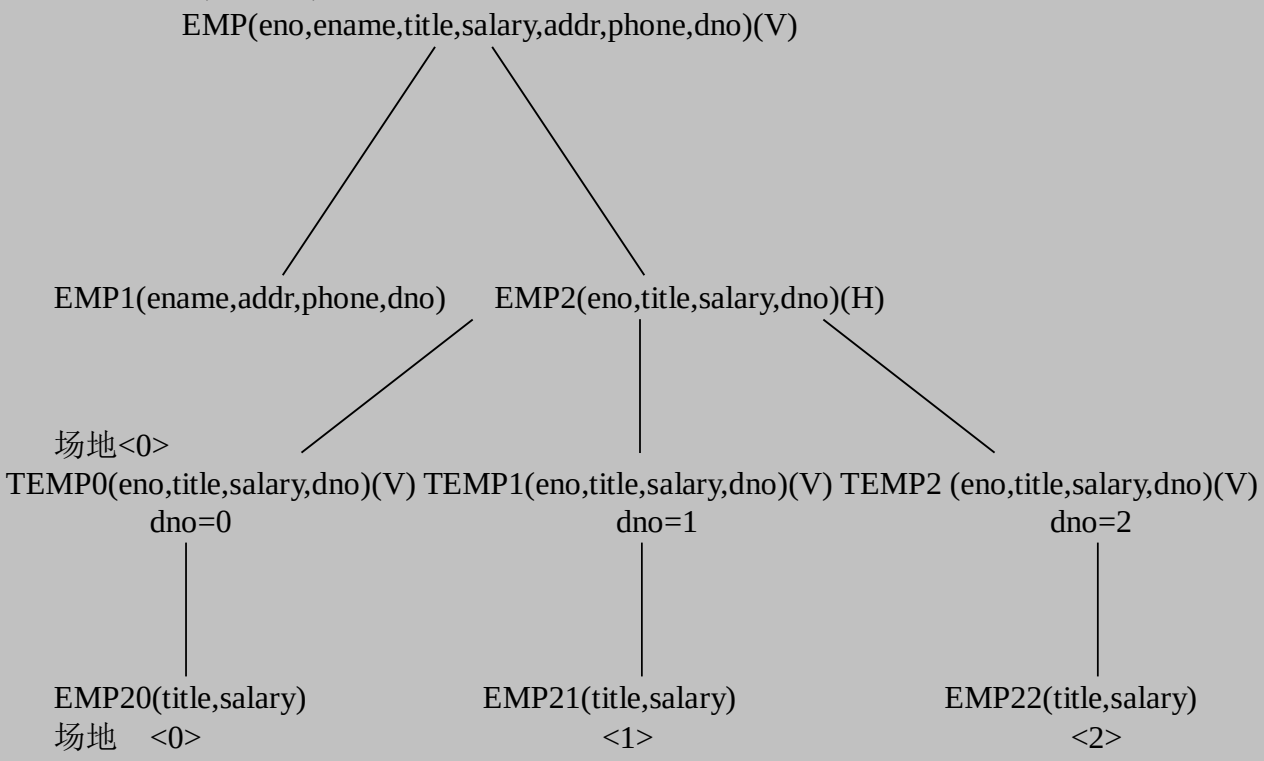
EMP22=$\Pi_{title,salary}$ TEMP2

Site EMP20(0), EMP21(1), EMP22(2);

3.2.2.指出各分片的类型，并画出分片树

EMP1,EMP2是对EMP进行的垂直分片。

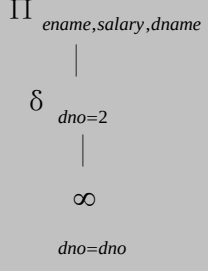
EMP20,EMP21,EMP22是对EMP2进行的水平分片。

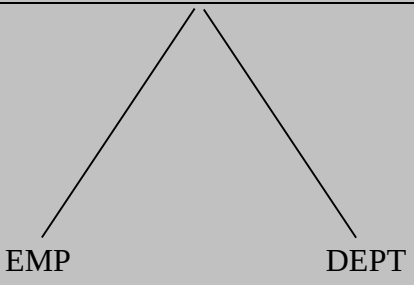


3.2.3.对查询 select ename,salary,dname from EMP,DEPT where dno=2.进行全局优化，画出优化后的全局查询树。

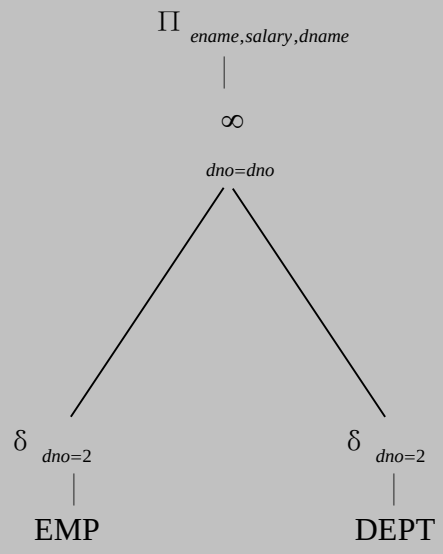
SQL语句 SELECT ename,salary,dname
FROM EMP,DEPT
WHERE dno=2

对应的关系代数为 $\Pi_{ename,salary,dname}(\delta_{dno=2}(EMP \infty DEPT))$



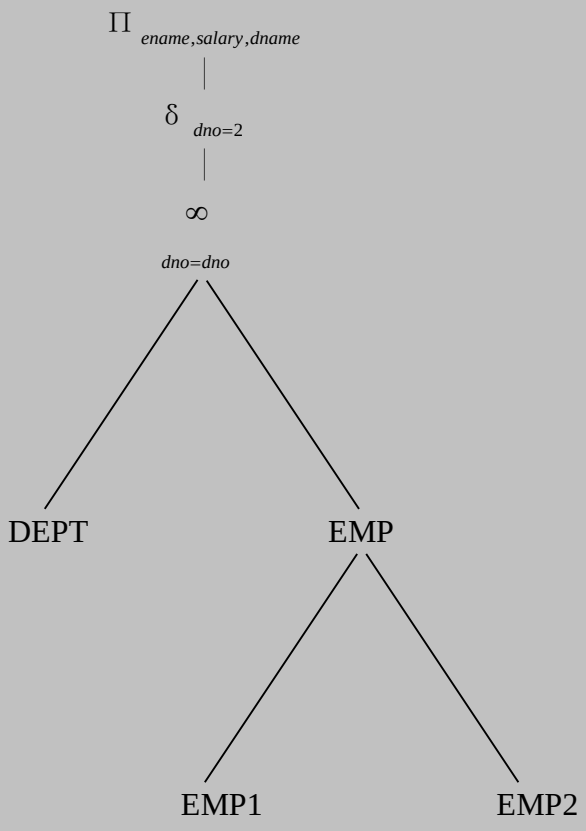


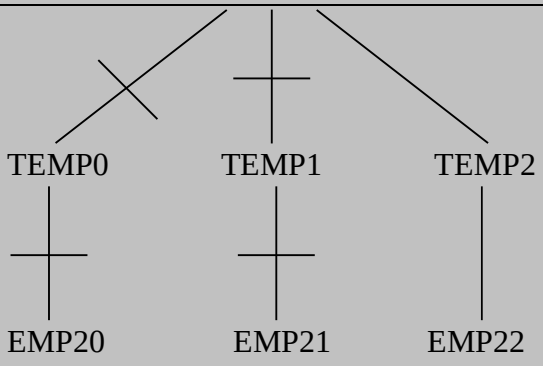
按C1、C2准则转换将一元操作下移



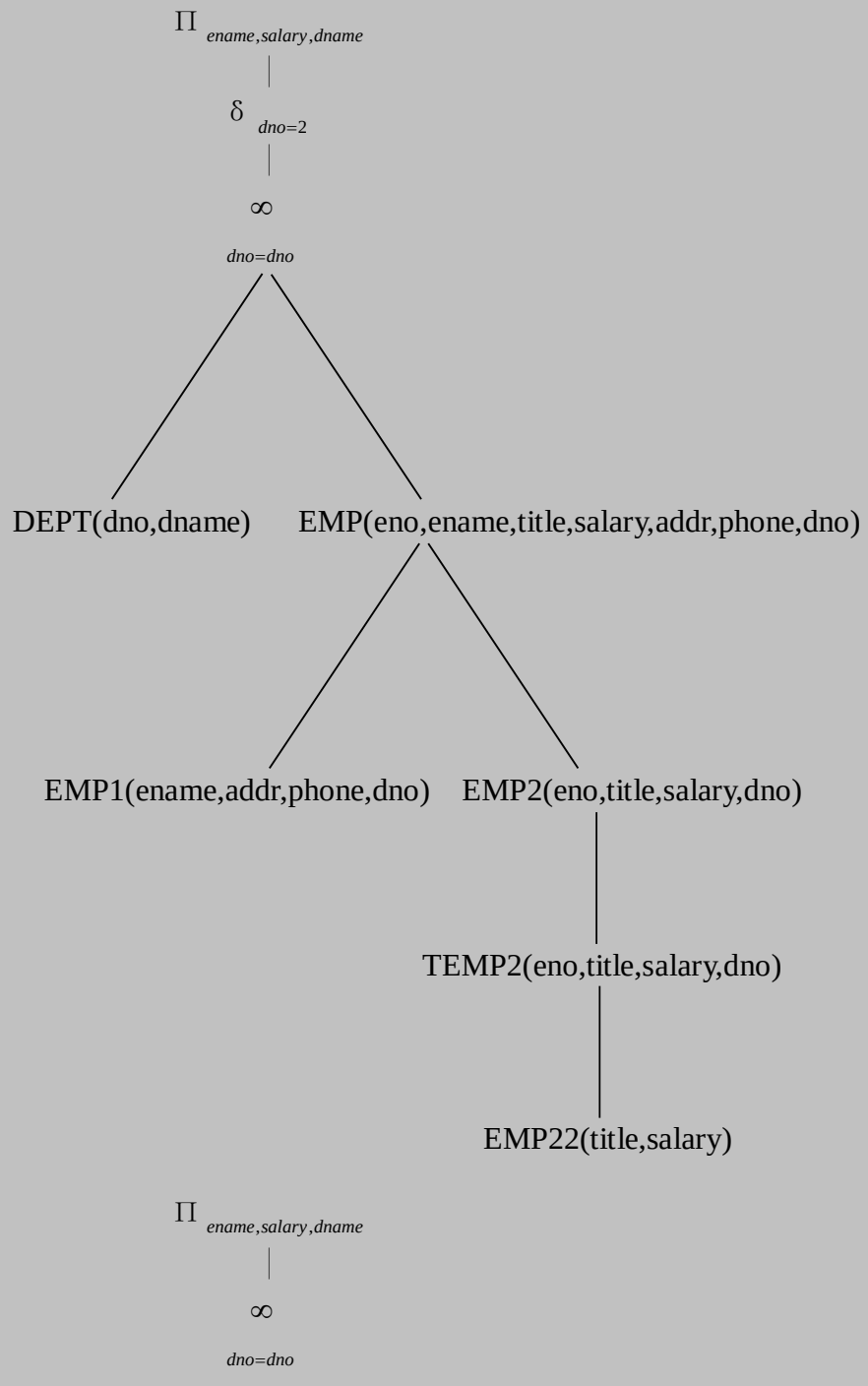
3.2.4.进行分片优化，画出优化后的分片查询树。

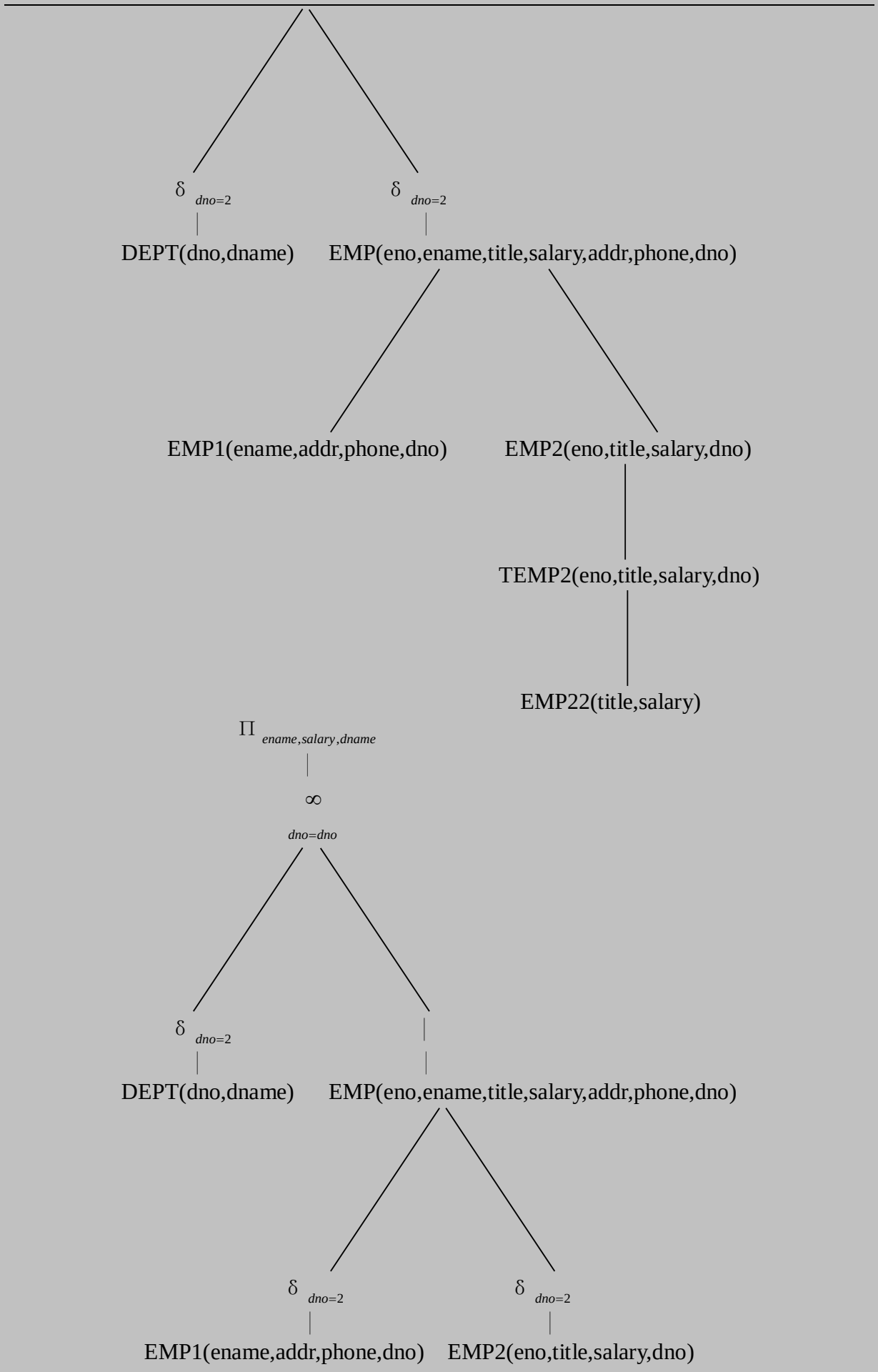
分解树的化简如图





去掉图中 TEMP0,TEMP1,EMP20,EMP21。




δ dno=2
δ dno=2
DEPT(dno,dname)
EMP(eno,ename,title,salary,addr,phone,dno)
EMP1(ename,addr,phone,dno)
EMP2(eno,title,salary,dno)
TEMP2(eno,title,salary,dno)
EMP22(title,salary)
Π ename,salary,dname
∞
dno=dno
δ dno=2
DEPT(dno,dname)
EMP(eno,ename,title,salary,addr,phone,dno)
δ dno=2
δ dno=2
EMP1(ename,addr,phone,dno)
EMP2(eno,title,salary,dno)

EMP22(title,salary)

3.3.对 3 个关系R,S和T的分布式连接，已知有如下的剖视图：

Card(R) = 1000 at 场地 S_1

	A	B
Length	20	10
Val	1000	1000

Card(S) = 4000 at 场地 S_2

	B	C
Length	10	5
Val	2000	100

Card(T) = 50 at 场地 S_3

	C
Length	5
Val	50

假设通信代价系数C_0= 0,C_1= 1;DOM(R,B)⊆DOM(S,B);DOM(T,C)⊆DOM(S,C)

3.3.1.按照SDD-1 半连接优化算法，逐步求出半连接优化集和最终执行场地；

分析剖视图得到连接图如下：

R---------- S ---------- T

B C

S_1 S_2 S_3

(连接图)

R ∞ S ∞ T

第一， 求可能的半连接集合

P1=S∝R

P2=S∝T

P3=R∝S

P4=T∝S

第二， 初始利益代价表

假设其中C_0=0, C_1=1；

计算选择度：

ρ_1（S∝R）=Val(R,B)/Val(S,B)=1000/2000=0.5。

其中Val(S,B)**是在关系**S**中属性**B**的不同值的个数也就是本题剖视图中的**Val**相的值，在这里是**2000。

同理Val(R,B)也是如此。

ρ_2（S∝T）=Val(T,C)/Val(S,C)=50/100=0.5。

其中Val(S,C)是在关系S中属性C的不同值的个数也就是本题剖视图中的Val相的值,这里是100。

同理Val(T,C)也是如此。

ρ_3（R∝S）=Val(S,B)/Val(R,B)=2000/1000=2≈1。

注意：当ρ>1时使ρ=1；

同理：ρ_4(T∝S)=1;

计算收益：

Benefit1(S∝R)=(1-ρ_1)*Size(S)*Card(S)*C_1=(1-0.5)*30*1000=15000；

其中Size(S)是关系S的宽度又称为元组的大小 ,Size(S)的值是S 的各个属性大小的和，这里是30。

其中Card(S)是关系S的所有元组的个数又称序数,这里是1000。

Benefit2(S∝T)= (1-ρ_2)*Size(S)*Card(S)*C_1=(1-0.5)*30*1000=15000；

Benefit3(E∝G)=0；

Benefit4(J∝G)=0；

计算代价(或称费用)：

Cost1(S∝R)=C_0+ C_I*Size(R,B)*Val(R,B)=10*1000=10000;

其中Size(R,B)是属性B的大小,这里是10。

其中Val(R,B)是属性B在关系R中的不同值的个数,这里是1000。

Cost2(S∝T)= C_0+ C_I*Size(T,C)*Val(T,C)=5*50=250

Cost3(R∝S)= +∞(无穷大)

Cost4(T∝S)= +∞(无穷大)

因为收益为0，所以代价很大，以至于不能计算，故视为+∞(无穷大)。

根据以上计算得到初始利益代价表，如下：

R∝S	选择性	收益	费用
P1:S $\underset{ENO}{SJ}$ R	1000/2000=0.5 根据R选择S的元组的比例	(1-0.5)*30*1000=15000 由于没选择S的部分的大小不用浪费通信代价,就是收益	10*1000=10000 传递R的部分的大小
P2:S $\underset{JNO}{SJ}$ T	50/100=0.5	(1-0.5)*30*1000=15000	5*50=250
P3:R $\underset{ENO}{SJ}$ S	2000/1000=2≈1	0	+∞
P4:T $\underset{JNO}{SJ}$ S	100/50=2≈1	0	+∞

根据初始的利益代价表，得到受益半连接集P={P1,P2}

第三， 选择半连接

1) 循环1

从受益半连接集P中选择代价最小的P2

令S' = S∝T, 得到关系S'{B,C}

连接图（R ∞ S' ∞ T）

R ---------- S' ---------- T

B C

S_1 S_2 S_3

A) 重新计算S'的概要图：

Card(S')= ρ_2 *Card(S)=0.5*4000=2000;

计算Val(S',B)

∵ B不属于选择谓词

Val(S,B)=2000

Val(S,B)/2=<Card(S')<2* Val(S,B)

故Val(S',B)=(Card(S')+ Val(S,B))/3=4000/3=1333

计算Val(S',C)

∵ C属于选择谓词

Val(S',C)= ρ_2 * Val(S,C)=50

因此，概要图如下：

Card(S')=2000

	B	C
Size	10	5
Val	1333	50

B) 重新求可能的半连接

P1= S' ∝R

(因为，P2=S∝T已经处理过，无需作P2= S' ∝T的半连接)

P3= R∝ S'

P4= T∝ S'

C) 重新计算利益代价

P1= S' ∝R

ρ_1=Val(R,B)/Val(S',B)= 1000/1333=0.75

Benefit1=(1-0.75)*Card(S')*Size(S') = 0.25*2000*15=7500

Cost3(S' ∝R) = C_I*Size(R,B)*Val(R,B)=10*1000=10000

P3= R∝ S'

ρ_3= Val(S',B)/Val(R,B)=1000/1000=1

Benefit3=(1- ρ_3)*Card(R)*Size(R)=0

Cost3(S' ∝T) = +∞

P4= T∝ S'

ρ_4= Val(S',C)/Val(T,C)=50/50=1;

Benefit4=(1-1)* Card(T)*Size(T)=0;

Cost4(T∝ S′) = +∞

利益代价表：

半连接∝	选择度ρ	利益Benefit	代价Cost
P1	0.75	7500	10000
P3	1	0	+∞
P4	1	0	+∞

所以受益的半连接集P={P1}

2) 循环2

从受益半连接集P中选择代价最小的P1

S″= S′∝R得：S″{B,C}

连接图 (R ∞ S″ ∞ T)

R---------- S″---------- T

B C

S_1 S_2 S_3

A) 重新计算 的概要图

Card(S″)= ρ_1*Card(S′)=0.75*2000=1500

计算Val(S″,B)

因为B属于选择谓词，所以

Val(S″,B)= ρ_3*Val(S′,B)=0.75*1000=750

计算Val(S″,C)

∵C不属于选择谓词

Val(S′,C)=50

Card(S″)>2* Val(S′,C)=100

故Val(S″,C)= Val(S′,C)=50

因此，概要图如下

Card(S″)=1500

	B	C
Size	10	5
Val	750	50

B) 重新求可能的半连接

（因为，P1= S′∝R已经处理过，无需作P1= S″∝R的半连接）

（因为，P2= S∝T已经处理过，无需作P2= S″∝T的半连接）

P3=R∝ S″

P4=T∝ S″

C) 重新计算利益代价

P3= R∝ S″

ρ_3=Val(S″,B)/Val(R,B)=750/1000=0.75

Benefit= Benefit3=(1- ρ_3)*Card(R)*Size(R)=0.25*1000*30=7500；

Cost3(R∝ S″)=C_1*Size(S″,B)*Val(S″,B)=10*750=7500；

P4= T∝ S''

ρ 4= Val(S'',C)/Val(T,C)=50/50=1;

Benefit4=0;

Cost4(T∝ S'')= +∞

利益代价表：

半连接∝	选择度ρ	利益Benefit	代价Cost
P3	0.75	7500	7500
P4	1	0	+∞

3) **循环3**

从受益半连接集P中选择代价最小的P3

R'= R∝ S''**得：** R'{A,B}

连接图 (R' ∞ S'' ∞ T)

R' ---------- S'' ---------- T

B C

S_1 S_2 S_3

A) **重新计算 的概要图**

Card(R')= ρ 3*Card(R)=0.75*1000=750

计算Val(R',A)

因为A不属于选择谓词， 又Val(R,A)=1000

Val(R,A)/2=< Card(R')<=2* Val(R,A)

所以

Val(R',A)=(Card(R')+ Val(R,A))/3=583

计算Val(R',B)

∵,B属于选择谓词

故Val(R',B)= ρ 3* Val(R,B)=0.75*1000=750

因此，概要图如下

Card(R')=750

	A	B
Size	20	10
Val	583	750

B) **重新求可能的半连接**

P1= S''∝R'

（**因为，** R'与T没有连接关系，故不用作P2 ）

P3= R'∝ S''

(P4=T∝ S''已经做过，不用再作。)

C) 重新计算利益代价

P1= $S'' \propto R'$

ρ_1=Val(R',B)/Val(S'',B)=750/750=1；

Benefit= Benefit3=(1-ρ_1)*Card(R)*Size(R)=0;

Cost3($S'' \propto R'$)=+∞；

P3= $R' \propto S''$

ρ_3= Val(S'',B)/Val(R',B)=750/750=1;

Benefit4=0;

Cost4($T \propto S''$)= +∞

利益代价表：

半连接∝	选择度ρ	利益Benefit	代价Cost
P3	1	0	+∞
P4	1	0	+∞

可知：无受益半连接，循环结束。

第四，计算执行场地

已知连接图：

R' ---------- S'' ---------- T

B C

S_1 S_2 S_3

概要图:

Card(R')=750

	A	B
Size	20	10
Val	583	750

Card(S'')=1500

	B	C
Size	10	5
Val	750	50

Card(T) = 50 at 场地 S_3

	C
Length	5
Val	50

根据连接图及概要图，代价计算:

Cost(S1)=Cost(S'')+Cost(T)

=Card(S'')*Size(S'')+Card(T)*Size(T)

=1500*15+50*5=22750=22.75K

Cost(S2)=Cost(R')+Cost(T)

=Card(R')*Size(R')+Card(T)*Size(T)

=750*30+50*5=22750=22.75K

Cost(S3)=Cost(R')+ Cost(S")

= Card(R')*Size(R')+Card(S")*Size(S")

=750*30+1500*15=45,000=45K

对比以上各执行场地的代价，可知：场地1,2的代价Cost(S1),Cost(S2)最小。

所以，最后执行场地选S1,S2都可以。

3.3.2.对以上结果做相应的优化处理。

在执行策略集中，消去用于缩减处于执行场地上的关系的半连接操作。

执行策略集P'={P1,P2,P3},其中P1,P2,P3分别为：

S' = S∝T=P1(在场地S2上执行)

S"= S'∝R=P2(在场地S2上执行)

R'= R∝ S"=P3(在场地S1上执行)

如果在场地S1上执行

因为：P3在场地S1上执行，可以从策略集合中消去P3

所以：P'={P1,P2}

因此：

总代价=Cost(S1)+Cost(P1)+Cost(P2)

= Cost(S1)+Size(T)*Card(T)+Size(R)*Card(R)

=22.75K+50*5+1000*10=33K

如果在场地S2上执行

因为：P1,P2在场地S2上执行，可以从策略集合中消去P1,P2。

所以：P'={P3}

因此：总代价=Cost(S1)+Cost(P3)

==Cost(S1)+Size(S")*Card(S")

=22.75K+1500*10=37.75K

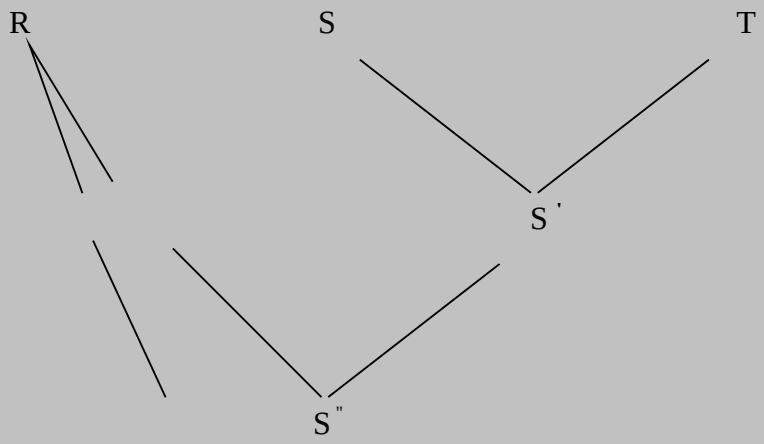
所以最终选择在“总代价”最低的场地S1上执行。

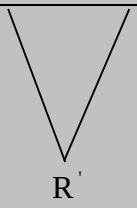
S'=S∝T

S"=S'∝R

R'=R∝ S"

如图





3.4. 用下面的关键字值的集合构造一颗B+树：（2,3,5,7,11,17,19,23,29,31）。假定树开始是空的，且关键字的值是以升序插入到B+树中去的，B+树每个节点中含的指针数为 4。

3.5. 考虑关系r_1(A,B,C)，r_2(C,D,E)，r_3(E,F)，假设不存在主关键字。设V(C, r_1)=900，V(C, r_2)=1100，V(E, r_2)=50,V(E, r_3)=100。设r_1有1000 个元组，r_2有1500个元组，r_3有750个元组。估算$r_1 \infty r_2 \infty r_3$的大小，并给出一种有效的执行策略。

3.6. 假设一个存储块中仅能存放一个记录且在内存中最多只有三个页框。请出在排序合并算法中每遍形成的Runs,排序属性为第一个属性：（kangaroo,17），(wallaby,21)，(emu,1)，(wombat,13)，(platypus,3)，(lion,8)，(warthg,4)，(zebra,11)，(meerkat,6)，(hornbill,2)，(baboon,12)。

4. 二零年春季试题

4.1.

4.1.1.分布库管理系统有哪些主要功能模块及其作用.

答：分布式数据库管理系统式对数据库进行管理和维护的一组软件，式分布式数据库系统的重要组成部分，是用户与分布式数据库系统的接口。它有三个组成部分。

全局数据库管理系统（GDBMS）：负责分布式数据库（DDB）中的全局数据库的连接、定位,策略面向全网的恢复能力。

局部数据库管理系统（LDBMS）：是各场地的数据库管理系统。

通信管理程序（CM）：保证分布式数据库系统中场地健信息传送的部分。

4.1.2.半连接方法和枚举法各适用于何种查询优化情况.

半联接技术缩减成连接操作的操作数，以降低通信费用。枚举法适用于缩减局部代价的情况。

查询操作的代价评估，需要综合考虑局部代价和传输代价。侧重于哪一方面，需根据系统组成环境确定。如侧重传输代价，局部代价可以忽略不计时，采用半联接技术较好；相反，如果侧重局部代价时，采用直接连接比采用半连接技术优越。因为直接连接技术实现简单，枚举法是基于直接连接的实现方法，此时应采用枚举法。

半连接优点：传输代价低。

半连接缺点：没有考虑局部代价；当选择“度交低”时，半连接技术才可行。

4.1.3.分布式事务有哪些基本性质.

原子性(Atomicity)、可串行性（一致性Consistency）、隔离性（Isolation）、持久性（Durability）。

这四个性质称为ACID性质，还有

执行特性：由一个控制进程（协调进程），协调各子事务的操作。
操作特性：通信原语，控制原语。
控制报文

4.1.4.什么是 2PC协议

答：为了实现分布式数据库事务的正确提交，一般采用两阶段提交协议（2PC）。2PC协议的基本思想是为全部的参与者做出提交或终止全部局部子事务的唯一决定。如果有一个参与者不能提交其子事务。则全部局部子事务终止。

2PC协议由两个阶段组成，第一阶段做出提交或终止全部子事务的决定，称为决定阶段；第二阶段实现第一阶段的决定，称为执行阶段。在日志部分丢失的情况下，2PC协议可以从多数故障中恢复出来。

协调者：在日志中写入“预提交”命令
　　发送“预提交”命令给所有参与者且开始计时
参与者：等待“预提交”命令
　　if 参与者可提交 then
　　　　begin
　　　　　在日志中写入子事务记录
　　　　　在日志中写入“准备提交”
　　　　　向协调者发“准备提交”信息
　　　　end
　　else　begin
　　　　　在日志中写入“abort”
　　　　　向协调者发“abort”信息
　　　　end
协调者：等待接收所有参与者的应答信息并检查限时。
　　If 超时或收到至少一个夭折 then
　　　　Begin
　　　　　在日志中记入“全部夭折”
　　　　　向所有子事务发“夭折”命令
　　　　end
　　else　begin
　　　　　在日志中记入“全局提交”
　　　　　向所有参与者发“提交”命令
　　　　end
参与者 等待协调者的命令
　　根据命令在日志中记入“夭折或提交”向协调者发送“执行”信息，
　　然后执行“提交”或“夭折”命令。
协调者 等待接收所有场地的“执行”信息
　　在日志中记入“完成”

4.2.下面是某个公司的人事关系数据库的全局模式：

EMP={ENO,ENAME,ADDR,POSITION,RANK,SAL,PHONE}

ENO表示职员号，ENAME表示职员姓名，ADDR表示住址，POSITION表示职务，RANK表示级别，SAL表示工资，PHONE表示电话号码，根据需要，要求将职工的个人信息（ENAME,ADDR,PHONE）与工资信息（POSITION,RANK,SAL）分别存放在场地1和场地2，但是，对于级别不低于“5”的工资信息，需另外存放在场地3。根据上述要

求，

4.2.1.将全局模式进行分片，写出分片定义和分片条件。

答：

$EMP1=\Pi_{ENO,ENAME,ADDR,PHONE,RANK}\ EMP$

$SAL1=\Pi_{POSITION,RANK,SAL}\ EMP$

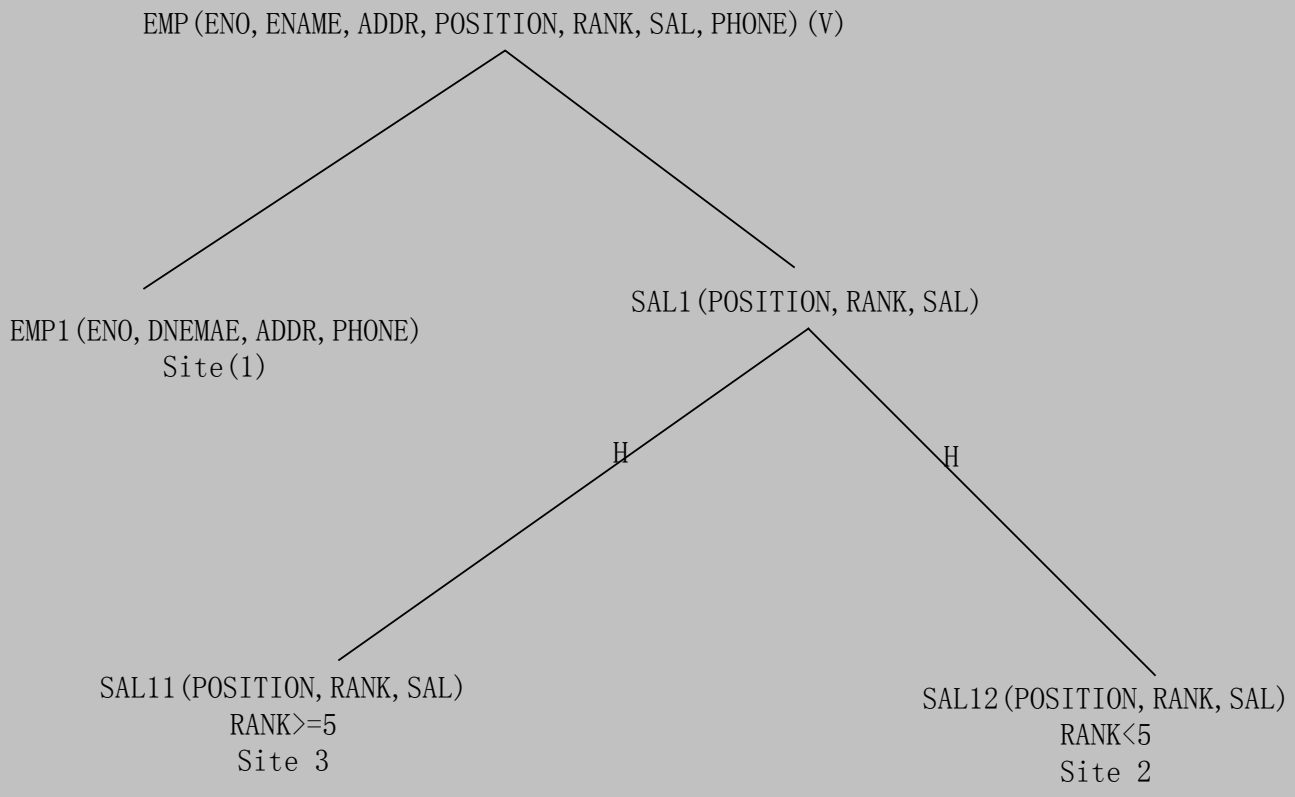
$SAL11=\delta_{RANK>="5"}\ SAL1$

$SAL12=\delta_{RANK<"5"}\ SAL1$

将EMP分成EMP1，SAL11，SAL12

Site　EMP1(1)　SAL11(3)　SAL12(2)

4.2.2.指出分片的类型，并画出“分片树”。



4.3.对题4.2所确定的分片模式，要求查询级别高于“6”的所有职员的姓名和工资，写出的在全局模式上的SQL查询语句，并要求转换成相应的关系代数表示，画出全局查询树。

4.3.1.进行全局优化，画出各步优化后的全局查询树。

答：

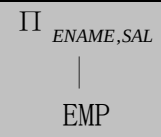
```
SELECT    ENAME,SAL
FROM      EMP
WHERE     RANK>“6”
```

用关系代数表示：　$\delta_{RANK>"6"}\ (\Pi_{ENAME,SAL}\ EMP)$

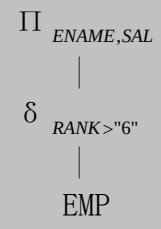
全局查询树如下

$\delta_{RANK>"6"}$

|



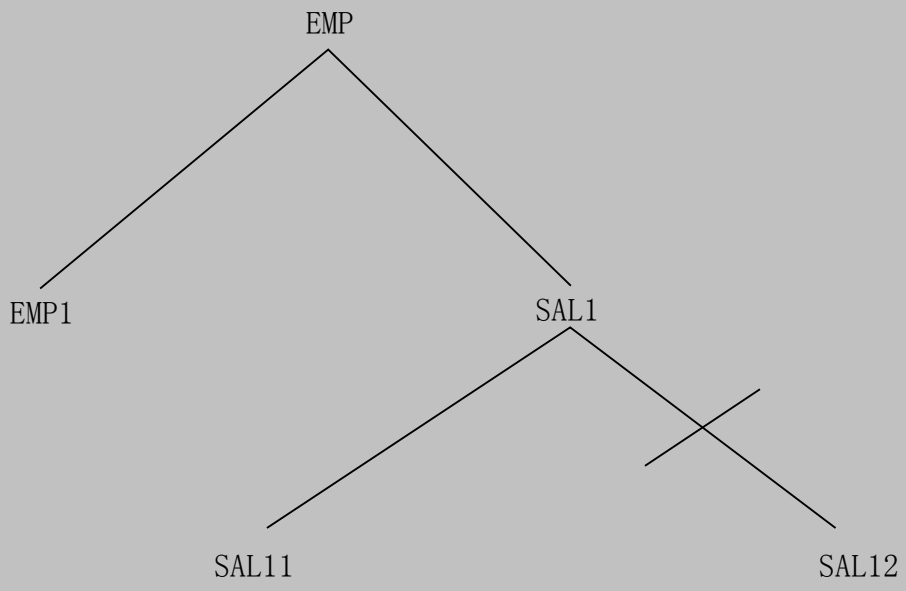
优化后的全局查询树如下



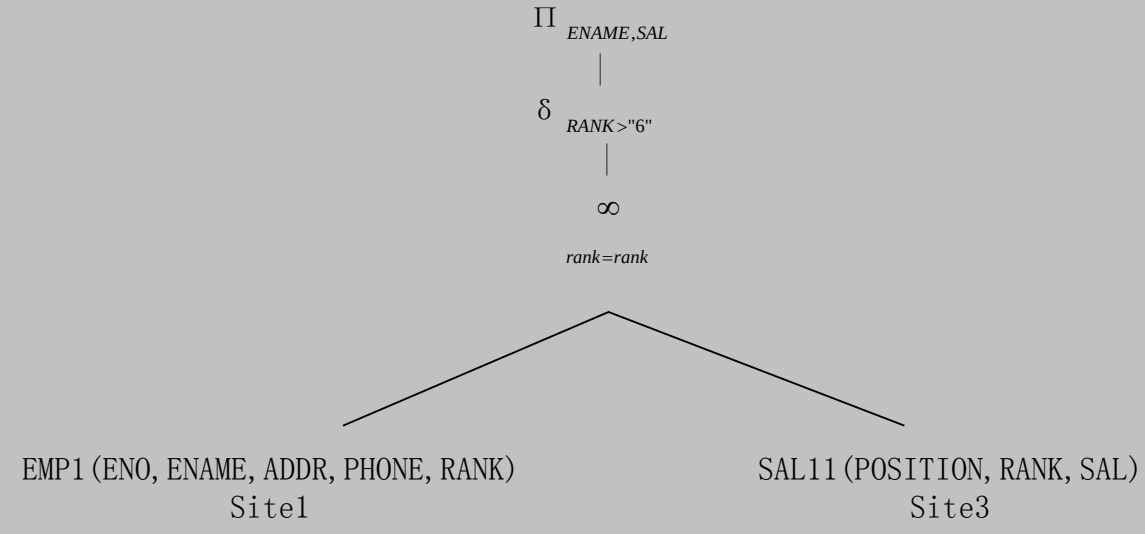
4.3.2.进行分片优化，画出各步优化后的分片查询树。

答：“分解树”化简

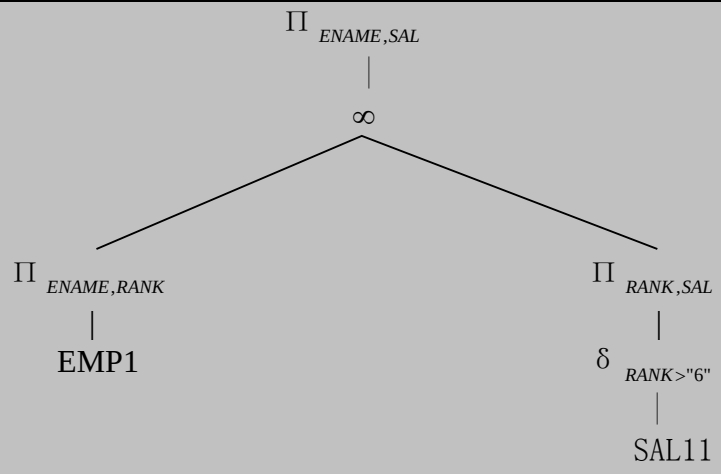
如图：



去掉图中 SAL12。



按C1、C2准则转换将一元操作下移。



4.4. 下面是一个数据库系统出现故障是，日志文件中记录的信息；

D_1^3	B_2	A_1	D_2^1	K	B_3	D_3^1	B_4	D_3^2	C_3	B_5	D_4^1

说明D_i^j D为数据记录，下标i表示事务号，上标j表示对数据库的第j步操作

根据上述信息，完成下面的处理：

4.4.1.找出发生故障时系统中的活动事务，确定出“反做”和“重做”事务集。

分析：T_1在检查点以前已经结束。故不用考虑。T_ε、T_4、T_5 在检查点前没开始不算作活动事务。

- 活动事务是指检查点上还没有结束答事务,即T_2。由于T_1在检查点上时已经A_1,故算作结束。初始化redo表和undo表，redo表={},undo表={ T_2 }
- 在日志中从检查点向前搜索，直到日志结尾。找出只有B记录没有C记录的事务，即在系统故障时未结束的事务，写入“反做事务表”。Undo表={ T_2 }∪{ T_4，T_5}={ T_2，T_4，T_5}
- 从检查点向前搜索，找出既有B记录又有C记录的事务，直到日志结尾,在系统故障时已经提交但部分结果未写入外存的事务，写入重做事务表。Redo表={ T_ε}

4.4.2.用C或其他语言定义出数据库记录（D记录）和检查点记录（K记录）的数据结构。

4.5. 设数据项x,y存放在S1 场地，u,v存放在S2 场地，有分布式事务T1 和T2,T1 在S1 场地的操作为R1(x)W1(x)R1(y)W1(y),T2 在S1 场地的操作为R2(x)R2(y)W2(y);T1 在 S2 场地上的操作作为R1(u)R1(v)W1(u),T2 在 S2 场地上的操作作为W2(u)R2(v)W2(v)。对下述 2 种情况，各举一例可能的局部历程（H1 和H2），并说明理由

串行调度	数据项x,y		数据项u,v	
Time	Site S1		Site S2	
|	T1	T2	T1	T2
|	R1(x)		R1(u)	
|	W1(x)		R1(v)	
|	R1(y)		W1(u)	
|	W1(y)			

	R2(x)	W2(u)
	R2(y)	R2(v)
↓	W2(y)	W2(v)

T1 < T2

4.5.1.局部分别是可串行化，而全局是不可串行化的

Time	数据项x,y Site S1		数据项u,v Site S2	
	T1	T2	T1	T2
	R1(x)			W2(u)
	W1(x)		R1(u)	R2(v)
	R1(y)	R2(x)		W2(v)
	W1(y)		R1(v)	
		R2(y)	W1(u)	
		W2(y)		
↓				

$T1^{S1} < T2^{S1}$　　$T2^{S2} < T1^{S2}$

T1、T2的所有子事务在每个站点都是可串行执行的。但

根据2PL协议事务T1在没有获得对v的锁之前是不会释放y的锁，而T2在没有获得y的锁之前是不会释放v的锁，T1和T2发生了死锁，故T1和T2之间在全局上是不可串行化的。

4.5.2.局部和全局都是可串行化的。要求按照严格的 2PL协议，加上适当的加锁和解锁命令，（注意，用rl(x)表示加读锁，wl(x)表示加对x加写锁，ul(x)表示解锁）

全局事务在全局范围内是可串行化的，必须是全局事务的所有子事务在每个局部站点上的可串行性在调度表中出现的顺序必须相同。即Ti^A<Tj^A,则对于所有拥有Ti和Tj代理的站点k都有Ti^k<Tj^k,

Time	数据项x,y Site S1		数据项u,v Site S2	
	T1	T2	T1	T2
	RL1(x)		RL1(u)	
	R1(x)	RL2(x)	R1(u)	WL2(u)
	WL1(x)	wait	RL1(v)	wait
	RL1(y)	.	R1(v)	.
	R1(y)	.	WL1(u)	.
	W1(y)	.	W1(u)	
	U1(x)	.	U1(u)	
	U1(y)	.	U1(v)	
		RL2(x)		WL2(u)
		R2(x)		W2(u)
		RL2(y)		RL2(v)
		R2(y)		R2(v)
		WL2(y)		WL2(v)
		W2(y)		W2(v)
		U2(x)		U2(u)
		U2(y)		U2(v)
↓				

5. 二零年秋试题

5.1.概念题

5.1.1.解释对象数据库系统中面向对象的相关概念

答：面向对象数据库系统是（Object Oriented Database System 简称OODBS）是数据库技术和面向对象技术程序设计方法相接合的产物。

对象表示数据库中面向对象的相关概念有：

5.1.1.1.对象与对象标识

答：现实世界的任一实体都被统一地模型化为一个对象。每个对象有一个唯一的标识，称为对象标识。

5.1.1.2.封装（encapsulation）

答：每个对象是其状态和行为的封装，其中状态是该对象一系列属性值的集合，而行为是在对象状态上的操作。操作也称为方法。

5.1.1.3.类

答：共享同样属性和方法集的所有对象构成了一个对象类（简称类）。一个对象是某个类的一个实例。

5.1.1.4.继承

答：可以定义一个类C的“子类”C1，C称为C1的“超类”（或“父类”）形成一个类的有限的层次结构，称为类层次。

5.1.1.5. 消息（Message）

答：由于对象是封装的，对象与外部的通信一般只能通过消息传递。

5.1.2.从概念上比较对象数据库模型与对象关系模型

答：面向对象数据库系统支持面向对象数据库模型（OO模型）。它用面向对象的观点来描述现实世界实体的逻辑组织、对象间的限制、联系等的模型。从面向记录上升为面向对象、面向具有复杂逻辑结构的一个整体、

对象－关系模型主要将关系数据库系统与面向对象数据库系统两方面的特征相接合。

面向对象数据库模型是面向对象程序设计范型在数据库系统中的改造，它的基础是将一个对象的相关数据和代码封装为一个单元，在概念上一个对象和系统其余部分的所有交互都要通过消息，因此对象和系统的其余部分的接口定义为一个所允许的消息的集合。

对象关系系统基于扩展关系模型的复杂数据与面向对象的一些概念（如对象标识和继承）结合在一起，对象关系数据模型扩展关系模型的方式是通过提供一个具有面向对象的更加丰富的类型系统，同时将一些成分加入到关系查询语言（如SQL）中以处理这些新增加的数据类型。

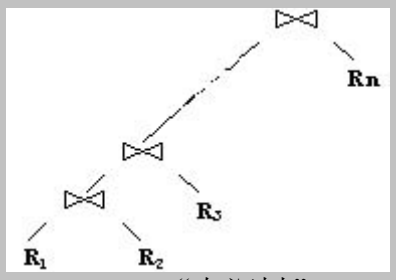
5.1.3.利用“左深树”、“右深树”、“浓密树”来进行查询优化的各自特点

1）单机与多机查询优化的区别。在研究查询优化时，并行数据库系统与单机数据库系统在下列方面有很大的区别：

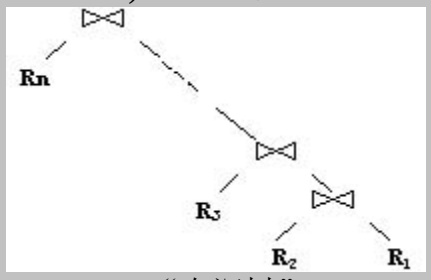
①优化目标。单机查询优化的目标是：对给定查询 Q，寻找一个具有最小工作总量的执行计划；多机并行查询优化的目标是：对给定查询 Q 寻找一个最小响应时间的执行计划。

②代价模型：它是优化执行计划的标准，它给出了估算执行计划的代价的方法。在单机中，查询的整体代价是各部分代价之和；而在多机查询中，整体代价为执行计划中，关键路径上各部分代价之和。

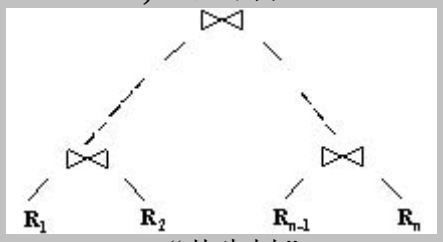
2）执行计划的表示。为了形象、直观，常用查询树表示执行计划。连接运算是并行数据库并行查询优化研究的重点对象。它的查询树有“左深树”（left-deep tree），“右深树”（right-deep tree）和“丛生树”（bushy tree）3 种。如图 9.13(a) 、b)、 c)。



a) “左深树”



b)“右深树”



c)“丛生树”

图 9.13 执行计划的查询树表示

“左深树”具有很好的流水线并行性。丛生树要消耗大量的计算资源。

5.1.4.试解释影响并行数据库系统中并行算法性能的三个因数

答：它们是CPU、I/O通道、通信线路。

5.1.5.简述用爬山算法进行查询优化的基本思想

设爬山者最初位于山上的某一点，目标是爬上峰顶。为此，爬山者可有4种走法，即向东、向南、向西、向北。爬山算法要求在每走一步之前，先计算分别向4个方向走一步后到达的新位置与原位置高度之差，即启发信息；然后根据这一信息决定向那个方向走。一般选择高度差最大的方向作为即将走步的方向，每走一步都要有经过计算得到的启发信息的引导，当到达某点时，若各个方向的高度差计算的结果都导致高度差下降，则认为该点就是峰顶，搜索结束。由于爬山算法每一步都是向梯度最陡的方向前进，而不是盲目攀登，因而可找到一条能很快到达山顶的路径。

5.2. 下面是某个公司一个人事关系数据库的全局模式：

EMP={ENO*,ENAME,POSITION,PHONE}　　PAY={POSITION*,SALARY}

ENO为职员号，POSITION为岗位。SALARY表示岗位对应的工资，*对应的属性表示主关键字。该公司分布在两个场地上，其中，在场地 1 经常处理所有职员数据，而场地 2 只处理工资低于 1000 的职员数据，为了节省磁盘空间和增大处理局部性：

5.2.1.将以上全局关系进行分片设计，写出分片定义和分片条件。

答：对关系PAY分片

PAY1= $\delta_{SALARY>=1000}$ PAY

PAY2= $\delta_{SALARY<1000}$ PAY

对关系EMP分片

EMP1=EMP $\underset{EMP.position=pay1.position}{\alpha}$ PAY1

EMP2=EMP $\underset{EMP.position=pay2.position}{\alpha}$ PAY2

分片定义和分片条件

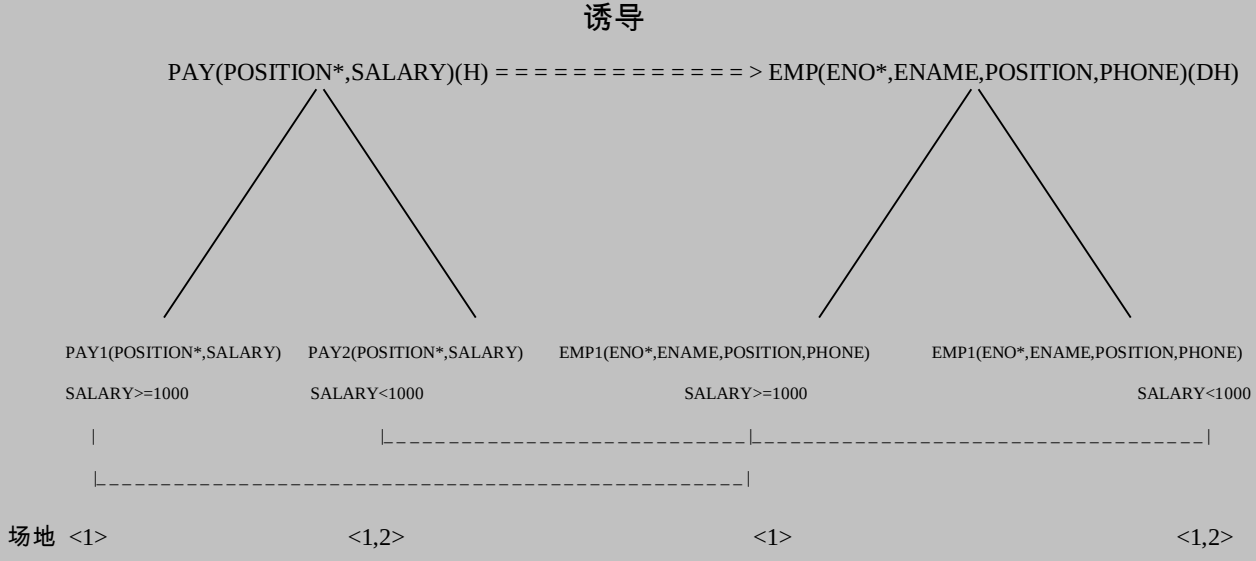
EMP1= $\Pi_{ENO,ENAME,POSITION,PHONE}$ (EMP $\underset{1.POSITION=2.POSITION}{\infty}$ ($\delta_{SALARY>=1000}$ PAY))

EMP2= $\Pi_{ENO,ENAME,POSITION,PHONE}$ (EMP $\underset{1.POSITION=1.POSITION}{\infty}$ ($\delta_{SALARY<1000}$ PAY))

5.2.2.指出分片的类型，并画出“分片树”。

对PAY(POSITION*,SALARY) 进行了水平分片。

对EMP（ENO*,ENAME,POSITION,PHONE）进行了依赖于PAY的诱导分片及分配。



5.2.3.给出分配设计。

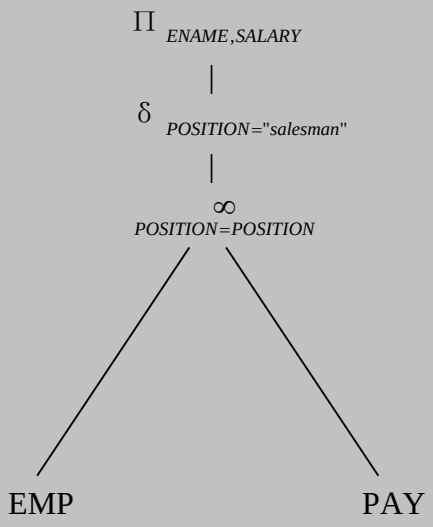
分配设计，如上图。

5.3.对题二所确定的分片模式，要求查询岗位为“salesman”的所有职员的姓名和工资，写出的在全局模式上的SQL查询语句，并要求转换成相应的关系代数表示，画出全局查询树。假设“salesman”的工资为800元。要求给出中间转换过程。

答：

```
SELECT    NAME,SALARY
FROM      EMP  JOIN  PAY  ON  EMP.POSITION = PAY.POSITION
WHERE     POSITION = “salesman”
```

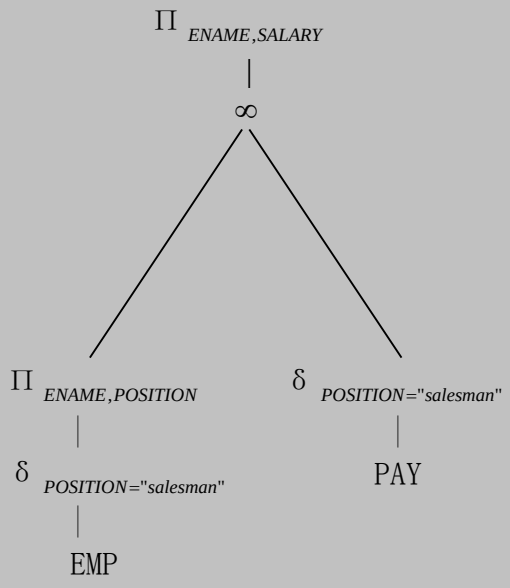
用关系代数表示：$\Pi_{ENAME,SALARY}(\delta_{POSITION="salesman"}((EMP \underset{POSITION=POSITION}{\infty} PAY))$



5.3.1.进行全局优化，画出优化后的全局查询树。

C1准则，缩减二元关系，用一元操作对二元操作的分配律将一元操作向下移。

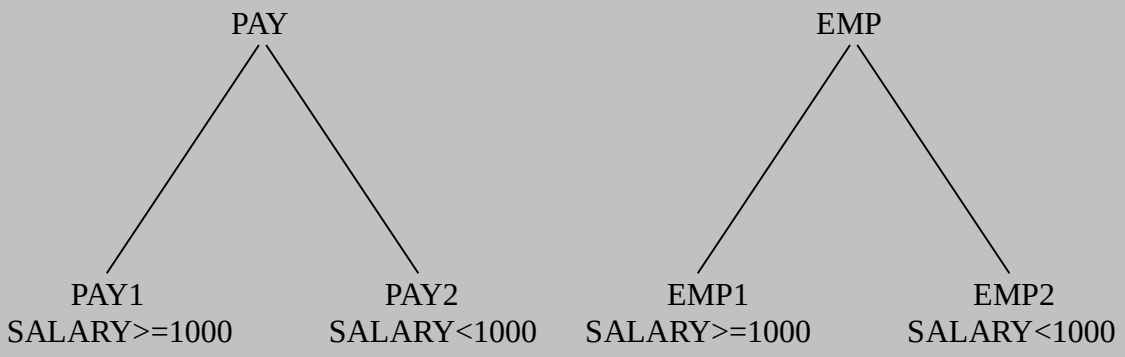
C2准则，缩减操作是关系。



这是优化后的全局查询数

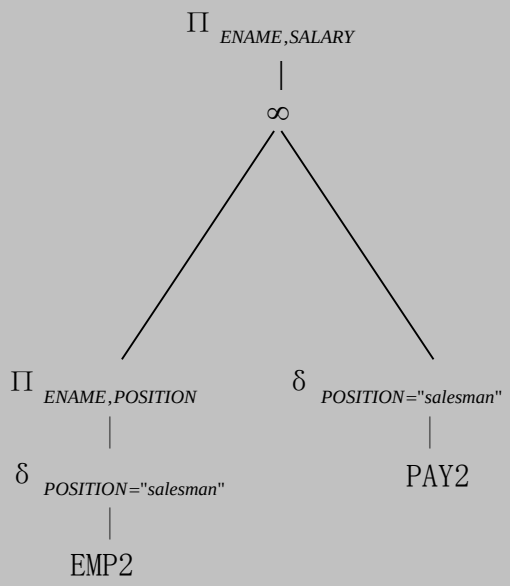
5.3.2.进行分片优化，画出优化后的分片查询树。

分解树化简

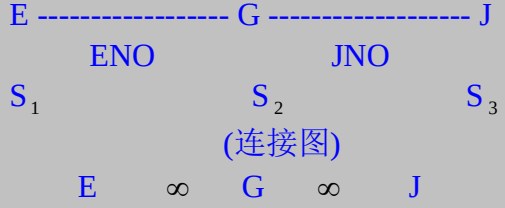


PAY1和EMP1不满足查询条件salesman的SALARY=800<1000.所以分解树分别化简为

优化后的分片查询树



5.4. 按如下给出的条件，求出半连接优化计划和执行场地，并作后优化处理



(连接图)

E ∞ G ∞ J

关系	序数Card	元组大小Size	属性	大小	特性值Val

E	500	50	E.ENO	4	500
G	4000	30	G.ENO	4	2000
			G.JNO	4	1000
J	200	40	J.JNO	4	200
已知：DOM(E.ENO)⊆DOM(G.ENO)　　DOM(J.JNO)⊆DOM(G.JNO)					

特性值：就是不同值的个数。序数：就是元组的个数。

答：

第一，可能的半连接集合

P1=G∝E

P2=G∝J

P3=E∝G

P4=J∝G

第二，初始利益代价表

假设其中C_0=0, C_I=1;

计算选择度：

ρ_1（G∝E）=Val(E,ENO)/Val(G,ENO)=500/2000=0.25。

其中Val(G,ENO)是在关系G中属性ENO的不同值的个数也就是本题所说的特性值。

同理Val(E,ENO)。

ρ_2（G∝J）=Val(J,JNO)/Val(G,JNO)=200/1000=0.2。

其中Val(G,JNO)是在关系G中属性JNO的不同值的个数也就是本题所说的特性值,这里是1000。

同理Val(J,JNO)。

ρ_3（E∝G）=Val(G,ENO)/Val(E,ENO)=2000/500=4。

注意：当ρ>1时使ρ=1;

同理：ρ_4(J∝G)=1;

计算收益：

Benefit1(G∝E)=(1-ρ_1)*Size(G)*Card(G)*C_1=(1-0.25)*30*4000=90000;

其中Size(G)是关系G的宽度又称为元组的大小，这里是30。

其中Card(G)是关系G的元组的个数又称序数,这里是4000。

Benefit2(G∝J)= (1-ρ_2)*Size(G)*Card(G)*C_1=(1-0.2)*30*4000=96000;

Benefit3(E∝G)=0;

Benefit4(J∝G)=0;

计算代价(或称费用)：

Cost1(G∝E)=C_0+ C_I*Size(E,ENO)*Val(E,ENO)=4*500=2000;

其中Size(E,ENO)是属性ENO的大小,这里是4。

其中Val(E,ENO)是属性ENO在关系E中的不同值的个数,这里是500。

Cost2(G∝J)=Size(J,JNO)*Val(J,JNO)=4*200=800

Cost3(E∝G)= +∞(无穷大)

Cost4(J∝G)= +∞(无穷大)

因为收益为0，所以代价很大，以至于不能计算，故+∞(无穷大)。

R∝S	选择性	收益	费用
P1:G $\underset{ENO}{SJ}$ E	500/2000=0.25 根据E选择G的元组的比例	(1-0.25)*30*4000=90000 由于没选择G的部分的大小,不用浪费通信代价就是收益	4*500=2000 传递E的部分的大小
P2:G $\underset{JNO}{SJ}$ J	200/1000=0.2	(1-0.2)*30*4000=96000	4*200=800
P3:E $\underset{ENO}{SJ}$ G	2000/2000=1	0	+∞
P4:J $\underset{JNO}{SJ}$ G	1000/1000=1	0	+∞

根据初始的利益代价表，得到受益半连接集P={P1,P2}

第三，选择半连接

4) 循环1

从受益半连接集P中选择代价最小的P2

令G' = G∝J,得到关系G'{ENO,JNO}

连接图（E ∞ G' ∞ J）

E ---------- G' ---------- J

ENO JNO

S_1 S_2 S_3

D) 重新计算G'的概要图：

Card(G')= ρ_2*Card(G)=0.2*4000=800;

Val(G',ENO)

∵ ENO不属于选择谓词

Val(G,ENO)=2000;

Card (G')<= Val(G,ENO)/2;

∴有：Val(G',ENO)= ρ_2* Val(G,ENO)=0.2*2000=400

Val(G',JNO)

∵ JNO属于选择谓词

∴ Val(G',JNO)= ρ_2* Val(G,JNO)= 0.2*1000=200

因此，概要图如下：

Card(G')=800

	ENO	JNO
Size	4	4
Val	400	200

E) 重新求可能的半连接

P1=G' ∝E

（因为，P2=G∝J已经处理过，无需作P2= G′∝J的半连接）

P3=E∝G′

P4=J∝G′

F) 重新计算利益代价

P1= G′∝E

ρ_1=Val(E,ENO)/Val(G′,ENO)= 500/400=1.25≈1

Benefit1=(1-1)*Card(G′)*Size(G′) = 0

Cost1=+∞

P3=E∝ G′

ρ_3= Val(G′,ENO)/Val(E,ENO)=400/500=0.8

Benefit3=(1-ρ_3)*Card(E)*Size(E)=0.2*500*50=5000

Cost3=Val(G′,ENO)*Size(G′,ENO)=400*4=1600

P4=J∝ G′

ρ_4=Val(G′,JNO)/Val(J,JNO)=200/200=1;

Benefit4=0;

Cost4=+∞

利益代价表：

半连接∝	选择度ρ	利益Benefit	代价Cost
P1	1	0	+∞
P3	0.8	5000	1600
P4	1	0	+∞

所以受益的半连接集P={P3}

5) 循环2

E′= E∝ G′得：E′{E.ENO}

连接图 (E′ ∞ G′ ∞ J)

E′---------- G′---------- J

ENO JNO

S_1 S_2 S_3

A) 重新计算概要图

Card(E′)=ρ_3*Card(E)=0.8*500=400

因为ENO属于选择谓词，所以

Val(E′,ENO)= ρ_3*Val(E,ENO)=0.8*500=400

因此，概要图如下

	ENO
Size	4
Val	400

B) 重新求可能的半连接

P1= G′∝ E′

（因为，P2=G∝J已经处理过，无需作P2= G′∝J的半连接）

（因为，P3= E∝G′已经处理过，无需作P2= E′∝ G′的半连接）

P4= J∝ G′

C) 重新计算利益代价

P1= G′∝ E′

ρ_1=Val(E′,ENO)/Val(G′,ENO)=400/400=1

Benefit=0;

Cost1=+∞

P4= J∝ G′

ρ_4= Val(G′,JNO)/Val(J,JNO)=200/200=1;

Benefit4=0;

Cost4=+∞

利益代价表：

半连接∝	选择度ρ	利益Benefit	代价Cost
P1	1	0	+∞
P4	1	0	+∞

可知：无受益半连接，循环结束。

第四，计算执行场地

已知连接图：

E′---------- G′---------- J

ENO JNO

S_1 S_2 S_3

概要图:

Card(E′)=400

	ENO
Size	4
Val	400

Card(G′)=800

	ENO	JNO
Size	4	4
Val	400	200

Card(J)=200

	JNO
Size	4
Val	200

根据连接图及概要图，代价计算:

Cost(S1)=Cost(G′)+Cost(J)

=Card(G′)* Size(G′)+Card(J)*Size(J)

=800*30+200*40=24000+8000=32000=32K

Cost(S2)=Cost(E')+Cost(J)
=Card(E')* Size(E')+Card(J)*Size(J)
=400*50+200*40=28K
Cost(S3)=Cost(E')+ Cost(G')
=Card(E')* Size(E')+Card(G')*Size(G')
=400*50+800*30=44K
对比以上各执行场地的代价，可知：场地2的代价Cost(S2)最小。
所以，最后执行场地选为2（S2）。

在执行策略集中，消去用于缩减处于执行场地上的关系的半连接操作。
执行策略集P'={P1,P2},其中P1,P2分别为：
G'=G∝J=P1(在场地S2上执行)
E'=E∝G'=P2(在场地S1上执行)
如果在场地S1上执行
因为：P2在场地S1上执行，可以从策略集合中消去P2
所以：P'={P1}
因此：
总代价=Cost(S1)+Cost(P1)
= Cost(S1)+Val(J.JNO)*Size(J.JNO)
=32K +200*4=32.8K

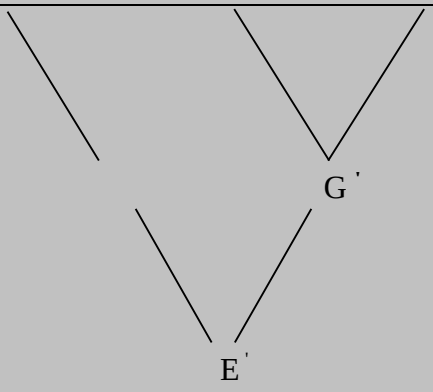
如果在场地S2上执行
因为：P1在场地S2上执行，可以从策略集合中消去P1。
所以：P'={P2}
因此：
总代价=Cost(S2)+Cost(P2)
= Cost(S2)+Val(G'.GNO)*Size(G'.GNO)
=28K +400*4=29.6K

如果在场地S3上执行
所以：P'={P1,P2}
因此：
总代价=Cost(S3)+Cost(P1)+Cost(P2)
= Cost(S2) +Val(J.JNO)*Size(J.JNO)+Val(G'.GNO)*Size(G'.GNO)
=44K+200*4+400*4=46.4K

所以最终选择在总代价最新的场地S2上执行。

G'=G∝J
E'=E∝G'
如图

E G J



5.5. 下面是当一个数据库系统出现故障时，日志文件中的信息

D_1^3	B_2	B_3	D_2^1	C_1	A_3	K	B_4	D_2^2	B_5	D_5^1	A_2	C_5	D_4^1	Crash

根据上述log 信息，完成下面的处理：

答：说明：

D_i^j:D为数据记录，下标i表示事务号，上标表示对数据的第j步操作；

B_i:表示事务i开始执行；

C_i:表示事务i提交；

A_i ：表示事务I废弃；

K：是检查点。

系统发生故障时，造成数据库不一致状态的原因为：

一些未完成的事务对数据库的更新已经写入外存数据库；一些已经提交的事务对数据库的更新还没写入外存。因此，基本的恢复操作分为两类，即“反做”（undo）和“重做”（redo）。“反做”也称撤消。

“反做”（undo）：是将一个数据项的值恢复到修改之前的值，即取消一个事务所完成的操作结果。当一个事务尚未提交时，如果缓冲区管理器允许该事务修改过的数据写到外存数据库，一旦此事务出现故障需要废弃时，就需对被这个事务修改过的数据项进行“反做”（undo），即恢复到其前像。“反做”(undo)保持数据库的原子性。

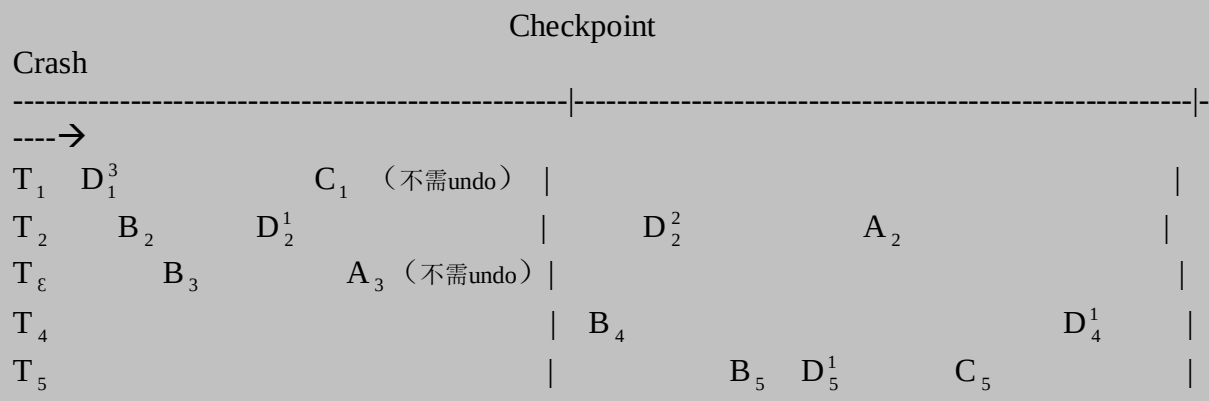
重做（redo）：是将一个数据项的值恢复到其修改后的值，即恢复一个事务的操作结果。当一个事务提交时，如果缓冲器允许该事务修改过答数据不立刻写到外存数据库，一旦事务出现故障，需对被这个事务修改过的数据项进行重做（redo），即恢复到其后像。“反做”(undo)保持数据库答永久性。

故障恢复的步骤：

- 恢复子系统首先从重启动文件中取得最近的检查点的地址，然后建立两个表：重做表和“反做表”。初始化两表，重做表初始状态为空；“反做表”初始状态为检查点上的活动事务。活动事务是指检查点上还没有结束答事务。
- 确定“反做事务表”（undo）：在日志中从检查点向前搜索，直到日志结尾。找出只有B记录没有C记录的事务，即在系统故障时未结束的事务，写入“反做事务表”。
- 确定重做事务表（redo）：从检查点向前搜索，找出既有B记录又有C记录的事务，直到日志结尾,在系统故障时已经提交但部分结果未写入外存的事务，写入重做事务表。
- “反做”（undo）“反做事务表”中的所有事务。根据日志,反向进行撤消操作，直到相应事务的B（Begin transaction）。

- 重做（redo）重做事务表中的所有事务，根据日志从检查点正向进行重做操作，直到相应的事务C（Commit transaction）

5.5.1.画出对应的事务并发执行图。



5.5.2.找出发生故障时系统中的活动事务，确定“反做”和“重做”事务集。

分析：T_1、T_ε在检查点以前已经结束。故不用考虑。T_4、T_5 在检查点前没开始不算作活动事务。

D_1^3	B_2	B_3	D_2^1	C_1	A_3	K	B_4	D_2^2	B_5	D_5^1	A_2	C_5	D_4^1	Crash

- 活动事务是指检查点上还没有结束答事务,即T_2。由于T_ε在检查点上时已经A_3,故算作结束。初始化redo表和undo表，redo表={},undo表={ T_2 }
- 在日志中从检查点向前搜索，直到日志结尾。找出只有B记录没有C记录的事务，即在系统故障时未结束的事务，写入“反做事务表”。Undo表={ T_2 }∪{ T_4 }={ T_2, T_4 }
- 从检查点向前搜索，找出既有B记录又有C记录的事务，直到日志结尾,在系统故障时已经提交但部分结果未写入外存的事务，写入重做事务表。Redo表={ T_5 }

5.5.3.指出需要undo的和redo的数据记录。

根据undo表和redo表

需要undo的数据记录为：D_2^2、D_4^1。

需要redo的数据记录为：D_5^1。

5.6. 设数据项x,y存放在S1 场地，u,v存放在S2 场地，有分布式事务T1 和T2。T1 在S1 场地的操作为R1(x)W1(x)R1(y)W1(y)，T2 在S1 场地的操作为R2(x)R2(y)W2(y)；T1 在S2 场地上的操作作为R1(u)R1(v)W1(u)，T2 在S2 场地上的操作作为W2(u)R2(v)W2(v)。对下述 2 种情况，各自举一种可能的局部历程（H1 和H2），如果是可串行化的，指出事务的执行次序。对第 3 种情况，给出符合基本 2PL协议的调度(T1 加锁命令用L1(X)表示，开锁命令U1(X)表示。对任何数据的加锁操作可在事务开始后立即进行)。

串行调度	数据项x,y		数据项u,v	
Time	Site	S1	Site	S2
|	T1	T2	T1	T2

	R1(x)		R1(u)	
	W1(x)		R1(v)	
	R1(y)		W1(u)	
	W1(y)			
		R2(x)		W2(u)
		R2(y)		R2(v)
↓		W2(y)		W2(v)

T1 < T2

5.6.1.局部是不可串行化的。

可串行化是指：让冲突的操作串行执行，非冲突的操作并行执行。

事务的并发调度要解决的最基本问题是并行执行事务对数据库答冲突操作如“ 读 - 写”(表示对同一个记录读的同时写,那么读到的数据不是最新数据)、“ 写 - 写”（同时对一个记录写）、“ 写 - 读”（对同一记录写的同时读，那么读到的不是写的结果，如下例）。换句话说，写操作不能和其他操作并行执行。

	数据项x,y		数据项u,v	
Time	Site S1		Site S2	
	T1	T2	T1	T2
|	R1(x)		R1(u)	W2(u)(读的同时写,冲突)
|	W1(x)	R2(x)(读到的x不是W1(x)写的结果,冲突)	R1(v)	R2(v)
|	R1(y)	R2(y)	W1(u)	W2(v)
|		W2(y)		
|	W1(y)			
↓				

5.6.2.局部是可串行化的，而全局是不可串行化的。

	数据项x,y		数据项u,v	
Time	Site S1		Site S2	
	T1	T2	T1	T2
|	R1(x)			W2(u)
|	W1(x)		R1(u)	R2(v)
|	R1(y)	R2(x)		W2(v)
|	W1(y)		R1(v)	
|		R2(y)	W1(u)	
|		W2(y)		
↓				

$T1^{S1} < T2^{S1}$　　$T2^{S2} < T1^{S2}$

T1、T2的所有子事务在每个站点都是可串行执行的。但

根据2PL协议事务T1在没有获得对v的锁之前是不会释放y的锁，而T2在没有获得y

的锁之前是不会释放v的锁，T1和T2发生了死锁，故T1和T2之间在全局上是不可串行化的。

5.6.3.局部是可串行化的，全局也是可串行化的。

全局事务在全局范围内是可串行化的，必须是全局事务的所有子事务在每个局部站点上的可串行性在调度表中出现的顺序必须相同。即Ti^A<Tj^A,则对于所有拥有Ti和Tj代理的站点k都有Ti^k<Tj^k,

	数据项x,y		数据项u,v	
Time	Site S1		Site S2	
↓	T1	T2	T1	T2
	RL1(x)		RL1(u)	
	R1(x)	RL2(x)	R1(u)	WL2(u)
	WL1(x)	wait	RL1(v)	wait
	RL1(y)	.	R1(v)	.
	R1(y)	.	WL1(u)	.
	W1(y)	.		
	U1(x)	.		
	U1(y)	.		
		RL2(x)		
		R2(x)		
		RL2(y)		
		R2(y)		
		WL2(y)		
		W2(y)		
		U2(x)		
		U2(y)		

5.7. 设计一种满足下列要求的索引结构。

5.7.1.被索引的数据集合为有序集

5.7.2.在有序集上的查询操作都是基于位置来进行的

5.7.3.当向“有序集”中插入或删除一个元素时，与该元素相关的后续元素的位置均要发生变化

5.7.4.元素的类型可为任意类型（这一个小问题的解决需要考虑语言的特征）

6. 二零一春季试题

6.1.

6.1.1.讨论集中式数据库和分布式数据库各自的优缺点。

数据库应用形态主要有二种：集中式数据库系统和分布式数据库系统。

集中式数据库系统的特点：将物理数据集中存放在主机上，由主机上的数据库管理系统统一管理整个数据库，用户可从终端上发出数据操作命令，经主机上的数据库管理系统接收处理后，再将所需数据送回终端。集中式数据库系统技术已成熟，数据共享能力、恢复能力较强，但所有数据库的操作均由主机管理，主机负荷较大，一旦主机故障，则导致系统全部瘫痪。

分布式数据库系统使各部门经常要用的数据能够就近存放，从而分散了工作负荷，

可靠性较高，局部发生故障不至于引起整个系统瘫痪。分布式数据库系统它在地理位置上是分布在一个含有多个数据库管理系统的计算机网络中，这些数据库管理系统构成分布式数据库管理系统，在分布式数据库管理系统中，每个用户所使用的数据可以不存储在自己使用的计算机上，而是由分布式数据库管理系统在机器之间通过网络从其它机器上传输过来。
分布式数据库：存取方便，速度快，但存在数据一致性和管理方面的问题；
集中式数据库：管理方便，但网络中信息流量大。

6.1.2.讨论在“局域网”和“广域网”两种情况下分布库设计的区别。

6.1.3.解释分片透明性、复制透明性和位置透明性等三级透明性的区别。

分片透明性是指用户不必关心数据的逻辑分片，不必关心数据物理位置分配的细节，也不必关心各个场地上数据库的数据模型。

复制透明性(数据的一致性) 指用户不必关心数据的一致性问题，不必关心数据在各个站点的一致性如何实现的细节，由分布数据库自动完成。

分布式数据库具有分布透明性，或称作位置透明性（Location Transparency）即程序的正确性不受数据从一个节点到另一个节点移动的影响，使用者自己并不需要知道数据是存在哪一个位置上。并且分布式数据库中个别位置或位置间通讯链发生故障时，也能继续运行，提供了比中央集权系统更高的可靠性。

分布式数据库的数据可以存储在靠近它正常使用的地方，可以减少响应时间与通讯费用。分布式数据库是将一些功能较少的数据库综合成一个功能更强的数据库系统，显然，增加一个新的站，远比用一个更大的系统代替已有的集中式系统要容易。使整个系统结构十分灵活，部分增减对系统的其它部分影响较小

6.1.4.解释 2PC协议如何在故障情况下保证事务的原子性的.

答 :为了实现分布式数据库事务的正确提交 ,一般采用两阶段提交协议(2PC)。2PC协议的基本思想是为全部的参与者做出提交或终止全部局部子事务的唯一决定。如果有一个参与者不能提交其子事务。则全部局部子事务终止。

2PC协议：一个分布事务涉及在几个不同结点上的操作，要在这些结点上建立执行这些操作的进程（该事务的代理进程），这些进程要相互通信，协调它们的操作进度（同步）。

在这些代理中，有一个根代理，在用户请求执行事务的结点（称为应用源点）建立，负责启动整个事务，负责发出命令：begin-transaction，commit，abort等。一个分布事务由该事务的根代理和其他结点的代理（称为子代理）组成。假设每个结点都有本地事务管理模块（LTM）可用，它保证本结点子事务（即子代理）的原子性，持久性等。

那么2PC协议把一个事务根代理结点的分布事务管理代理（DTM）称为协调者，其他结点的子代理的分布事务管理代理DTM称为参与者。由协调者指挥分布事务的交付过程。

2PC协议由两个阶段组成，第一阶段做出提交或终止全部子事务的决定，称为决定阶段；第二阶段实现第一阶段的决定，称为执行阶段。在日志部分丢失的情况下，2PC协议可以从多数故障中恢复出来。

协调者：在日志中写入“ 预提交” 命令

发送“ 预提交” 命令给所有参与者且开始计时

参与者：等待“ 预提交” 命令

if 参与者可提交 then

begin

在日志中写入子事务记录

在日志中写入“ 准备提交”

向协调者发“ 准备提交” 信息

end

else begin

在日志中写入“ abort”

向协调者发“ abort” 信息

end

协调者：等待接收所有参与者的应答信息并检查限时。

If 超时或收到至少一个夭折 then

Begin

在日志中记入“ 全部夭折”

向所有子事务发“ 夭折” 命令

end

else begin

在日志中记入“ 全局提交”

向所有参与者发“ 提交” 命令

end

参与者 等待协调者的命令

根据命令在日志中记入“ 夭折或提交” 向协调者发送“ 执行” 信息，然后执行“ 提交” 或“ 夭折” 命令。

协调者 等待接收所有场地的“ 执行” 信息

在日志中记入“ 完成”

6.1.5.解释严格 2PL协议与基本 2PL协议的区别

两段锁协议（2PL-TWO-Phase Lock protocol）
在两段锁协议中，一个事务可以分为两个阶段：
①扩展阶段：可以加新的锁，但不能解锁；
②收缩阶段：可以解锁，但不能加新的锁。
根据解锁的时机，又可分为一般2PL和严格2PL，后者只能在事务终点解锁
在分布事务管理中，两段封锁（2PL）所强制的事务的并发调度仍然是可串行化的。但为了保证隔离性，要在事务交付后才释放全部封锁（严格2PL）。

6.2. 下面是某个公司一个人事关系数据库的全局模式：

EMP={ENO*,ENAME,POSITION,PHONE}　　PAY={POSITION*,SALARY}

ENO为职员号，POSITION为岗位。SALARY表示岗位对应的工资，*对应的属性表示主关键字。该公司分布在两个场地上，其中，在场地 1 经常处理所有职员数据，而场地 2 只处理工资低于 1000 的职员数据，为了节省磁盘空间和增大处理局部性：

6.2.1.将以上全局关系进行分片设计，写出分片定义和分片条件。

答：对关系PAY分片

$PAY1=\delta_{SALARY>=1000}PAY$

$PAY2=\delta_{SALARY<1000}PAY$

对关系EMP分片

$EMP1=EMP \underset{EMP.position=pay1.position}{\alpha} PAY1$

$EMP2=EMP \underset{EMP.position=pay2.position}{\alpha} PAY2$

分片定义和分片条件

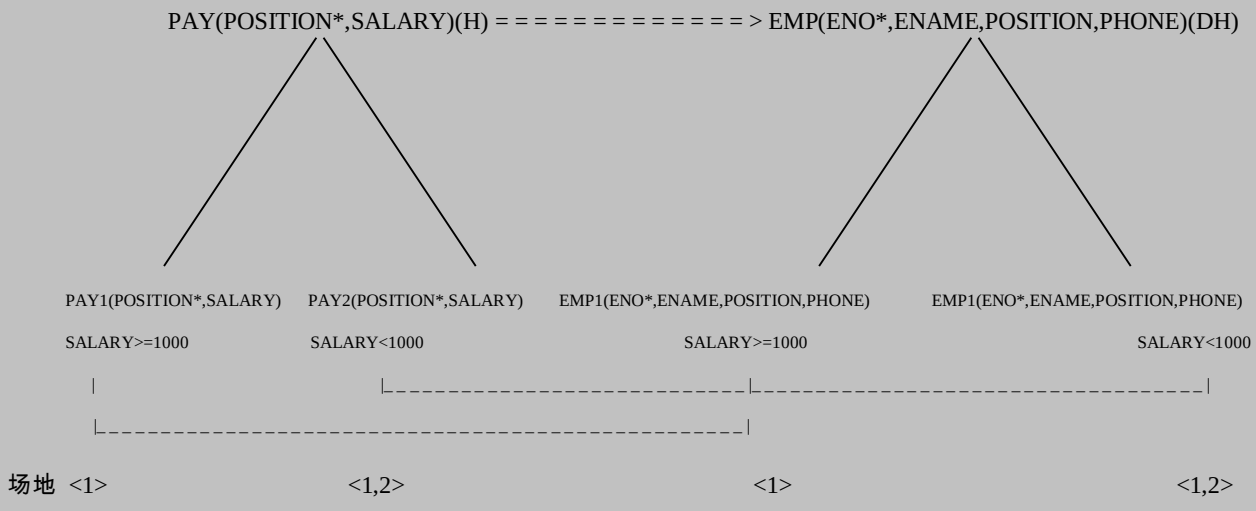
$EMP1=\Pi_{ENO,ENAME,POSITION,PHONE}(EMP \underset{1.POSITION=2.POSITION}{\infty}(\delta_{SALARY>=1000}PAY))$

$EMP2=\Pi_{ENO,ENAME,POSITION,PHONE}(EMP \underset{1.POSITION=1.POSITION}{\infty}(\delta_{SALARY<1000}PAY))$

6.2.2.指出分片的类型，并画出“分片树”。

对PAY(POSITION*,SALARY) 进行了水平分片。

对EMP（ENO*,ENAME,POSITION,PHONE）进行了依赖于PAY的诱导分片及分配。



6.2.3.给出分配设计。

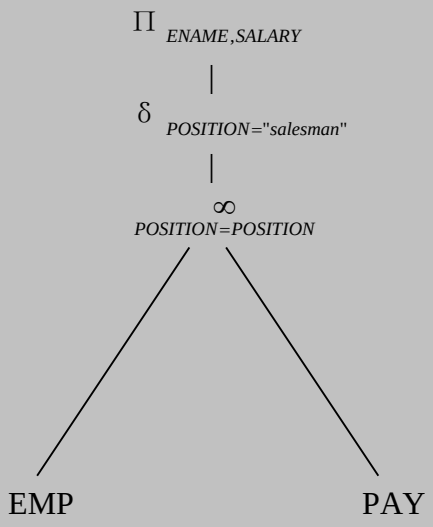
分配设计，如上图。

6.3. 对题二所确定的分片模式，要求查询岗位为“salesman”的所有职员的姓名和工资，写出的在全局模式上的SQL查询语句，并要求转换成相应的关系代数表示，画出全局查询树。假设“salesman”的工资为 1500 元。要求给出中间转换过程。

答：

```
SELECT    NAME,SALARY
FROM      EMP   JOIN   PAY ON   EMP.POSITION = PAY.POSITION
WHERE     POSITION =    “salesman”
```

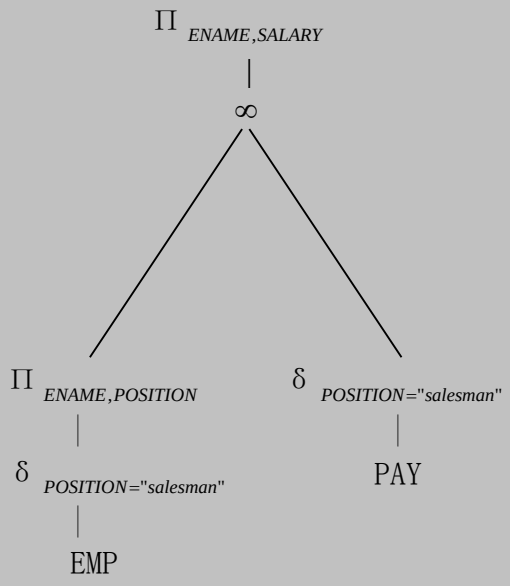
用关系代数表示：$\Pi_{ENAME,SALARY}(\delta_{POSITION="salesman"}((EMP \underset{POSITION=POSITION}{\infty} PAY))$



6.3.1.进行全局优化，画出优化后的全局查询树

C1准则，缩减二元关系，用一元操作对二元操作的分配律将一元操作向下移。

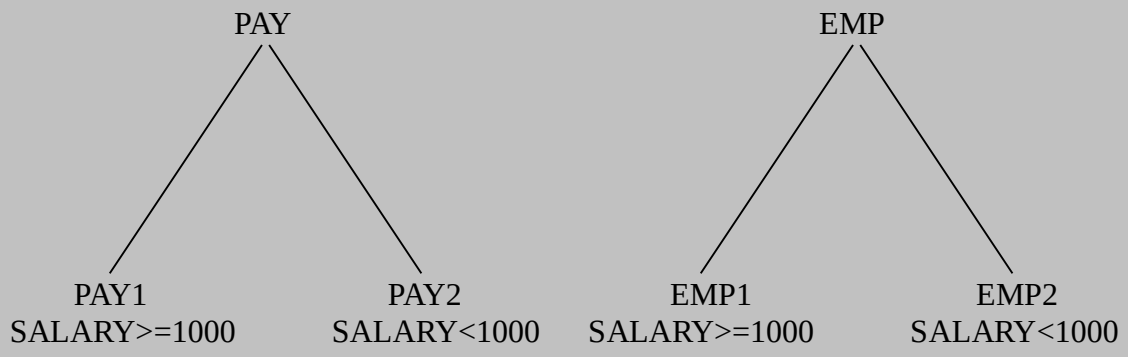
C2准则，缩减操作是关系。



这是优化后的全局查询数

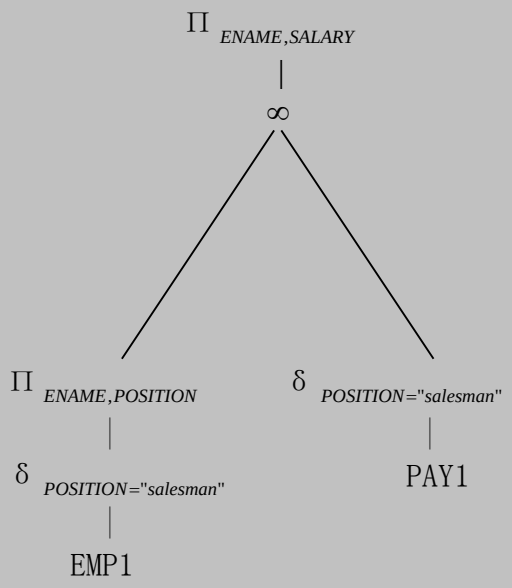
6.3.2.进行分片优化，画出优化后的分片查询树。

分解树化简



PAY2和EMP2不满足查询条件salesman的SALARY=1500>1000.所以分解树分别化简为

优化后的分片查询树



6.4. 下面是当一个数据库系统出现故障时，日志文件中的信息

D_1^3	B_2	B_3	D_2^1	C_1	A_3	K	B_4	D_2^2	B_5	D_5^1	A_2	C_5	D_4^1	Crash

根据上述log 信息，完成下面的处理：

答：说明：

D_i^j:D为数据记录，下标i表示事务号，上标表示对数据的第j步操作；

B_i:表示事务i开始执行；

C_i:表示事务i提交；

A_i ：表示事务i废弃；

K：是检查点。

系统发生故障时，造成数据库不一致状态的原因为：

一些未完成的事务对数据库的更新已经写入外存数据库；一些已经提交的事务对数据库的更新还没写入外存。因此，基本答恢复操作分为两类，即“反做”（undo）和重做（redo）。“反做”也称撤消。

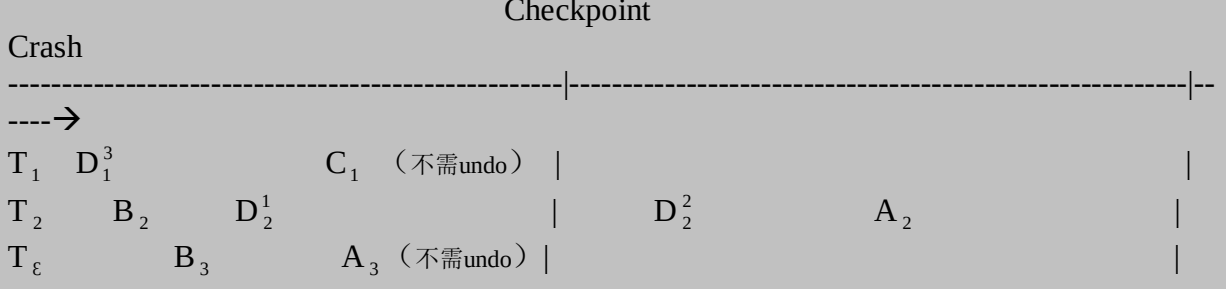
“反做”（undo）是将一个数据项的值恢复到修改之前的值，即取消一个事务所完成的操作结果。当一个事务尚未提交时，如果缓冲区管理器允许该事务修改过的数据写到外存数据库，一旦此事务出现故障需要废弃时，就需对被这个事务修改过的数据项进行“反做”（undo），即恢复到其前像。“反做”(undo)保持数据库的原子性。

重做（redo）是将一个数据项的值恢复到其修改后的值，即恢复一个事务的操作结果。当一个事务提交时，如果缓冲器允许该事务修改过的数据不立刻写到外存数据库，一旦事务出现故障，需对被这个事务修改过的数据项进行重做（redo），即恢复到其后像。“反做”(undo)保持数据库答永久性。

故障恢复的步骤：

- 恢复子系统首先从重启动文件中取得最近的检查点的地址，然后建立两个表：“重做表”和“反做表”。初始化两表，重做表的初始状态为空；“反做表”的初始状态为检查点上的活动事务。活动事务是指检查点上还没有结束答事务。
- 确定“反做事务表”（undo）：在日志中从检查点向前搜索，直到日志结尾。找出只有B记录没有C记录的事务，即在系统故障时未结束的事务，写入“反做事务表”。
- 确定重做事务表（redo）：从检查点向前搜索，找出既有B记录又有C记录的事务，直到日志结尾,在系统故障时已经提交但部分结果未写入外存的事务，写入重做事务表。
- “反做”（undo）“反做事务表”中的所有事务。根据日志，反向进行撤消操作，直到相应事务的B（Begin transaction）。
- 重做（redo）重做事务表中的所有事务，根据日志从检查点正向进行重做操作，直到相应的事务C（Commit transaction）

6.4.1.画出对应的事务并发执行图。



T_4	|	B_4			D_4^1	|
T_5	|		B_5	D_5^1 C_5		|

6.4.2.找出发生故障时系统中的活动事务，确定出“反做”和重做事务集。

分析：T_1、T_6在检查点以前已经结束。故不用考虑。T_4、T_5 在检查点前没开始不算作活动事务。

- 活动事务是指检查点上还没有结束答事务,即T_2。由于T_6在检查点上时已经A_3,故算作结束。初始化redo表和undo表，redo表={},undo表={ T_2 }
- 在日志中从检查点向前搜索，直到日志结尾。找出只有B记录没有C记录的事务，即在系统故障时未结束的事务，写入“反做事务表”。Undo表={ T_2 }∪{ T_4 }={ T_2, T_4 }
- 从检查点向前搜索，找出既有B记录又有C记录的事务，直到日志结尾,在系统故障时已经提交但部分结果未写入外存的事务，写入重做事务表。Redo表={ T_5 }

6.4.3.指出需要undo的和redo的数据记录。

根据undo表和redo表
需要undo的数据记录为：D_2^2、D_4^1。
需要redo的数据记录为：D_5^1。

6.5. 设数据项x,y存放在S1 场地，u,v存放在S2 场地，有分布式事务T1 和T2,T1 在S1 场地的操作为 R1(x)W1(x)R1(y)W1(y),T2 在 S1 场地的操作为R2(x)R2(y)W2(y);T1 在S2 场地上的操作作为R1(u)R1(v)W1(u),T2 在S2 场地上的操作作为W2(u)R2(v)W2(v)。对下述 2 种情况，各举一例可能的局部历程（H1和H2），如果是可串行化的，指出事务的执行次序。对第 3 种情况，给出符合基本 2PL协议的调度。（T1 加锁命令用L1(X)表示，开锁命令U1(X)表示。对任何数据的加锁可以在事务开始后立即进行）。

串行调度	数据项x,y		数据项u,v	
Time	Site S1		Site S2	
|	T1	T2	T1	T2
|	R1(x)		R1(u)	
|	W1(x)		R1(v)	
|	R1(y)		W1(u)	
|	W1(y)			
|		R2(x)		W2(u)
|		R2(y)		R2(v)
↓		W2(y)		W2(v)

T1 < T2

6.5.1.局部是不可串行化的。

可串行化是指：让冲突的操作串行执行，非冲突的操作并行执行。

事务的并发调度要解决的最基本问题是并行执行事务对数据库答冲突操作如 读-写(表示对同一个记录读的同时写,那么读到的数据不是最新数据)、写-写（同时对一个记录写）、写-读（对同一记录写的同时读，那么读到的不是写的结果，如下例）。换句话说，写操作不能和其他操作并行执行。

	数据项x,y		数据项u,v	
Time	Site S1		Site S2	
|	T1	T2	T1	T2
|	R1(x)		R1(u)	W2(u)(读的同时写,冲突)
|	W1(x)	R2(x)(读到的x不是W1(x)写的结果,冲突)	R1(v)	R2(v)
|	R1(y)	R2(y)	W1(u)	W2(v)
|		W2(y)		
|	W1(y)			
↓				

6.5.2.局部是可串行化的，而全局是不可串行化的。

	数据项x,y		数据项u,v	
Time	Site S1		Site S2	
|	T1	T2	T1	T2
|	R1(x)			W2(u)
|	W1(x)		R1(u)	R2(v)
|	R1(y)	R2(x)		W2(v)
|	W1(y)		R1(v)	
|		R2(y)	W1(u)	
|		W2(y)		
↓				

$T1^{S1} < T2^{S1}$　　$T2^{S2} < T1^{S2}$

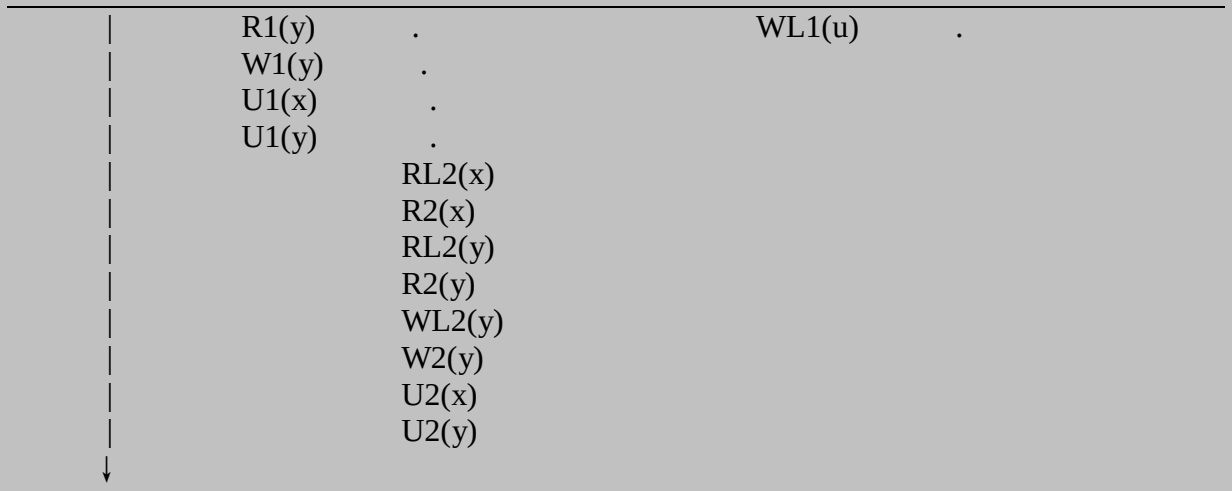
T1、T2的所有子事务在每个站点都是可串行执行的。但

根据2PL协议事务T1在没有获得对v的锁之前是不会释放y的锁，而T2在没有获得y的锁之前是不会释放v的锁，T1和T2发生了死锁，故T1和T2之间在全局上是不可串行化的。

6.5.3.局部是可串行化的，全局也是可串行化的。

全局事务在全局范围内是可串行化的，必须是全局事务的所有子事务在每个局部站点上的可串行性在调度表中出现的顺序必须相同。即$Ti^A < Tj^A$,则对于所有拥有Ti和Tj代理的站点k都有$Ti^k < Tj^k$,

	数据项x,y		数据项u,v	
Time	Site S1		Site S2	
|	T1	T2	T1	T2
|	RL1(x)		RL1(u)	
|	R1(x)	RL2(x)	R1(u)	WL2(u)
|	WL1(x)	wait	RL1(v)	wait
|	RL1(y)	.	R1(v)	.


R1(y)
W1(y)
U1(x)
U1(y)
WL1(u)
RL2(x)
R2(x)
RL2(y)
R2(y)
WL2(y)
W2(y)
U2(x)
U2(y)