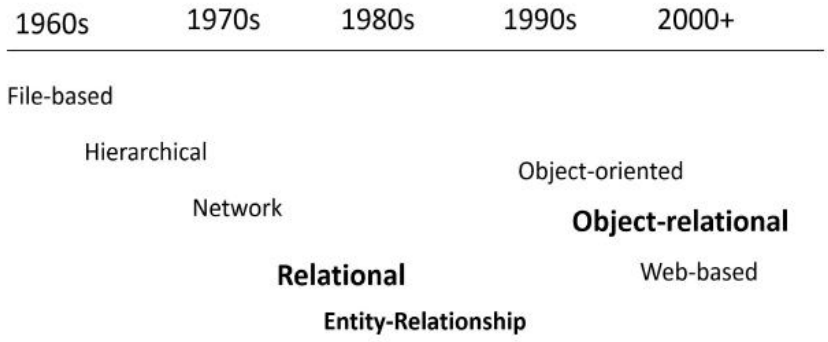
高级数据库技术

- n 历史
- n 采用的数据模型
- n 查询语言
- n 原理与技术
- n 功能
- n 体系结构
- n 特点

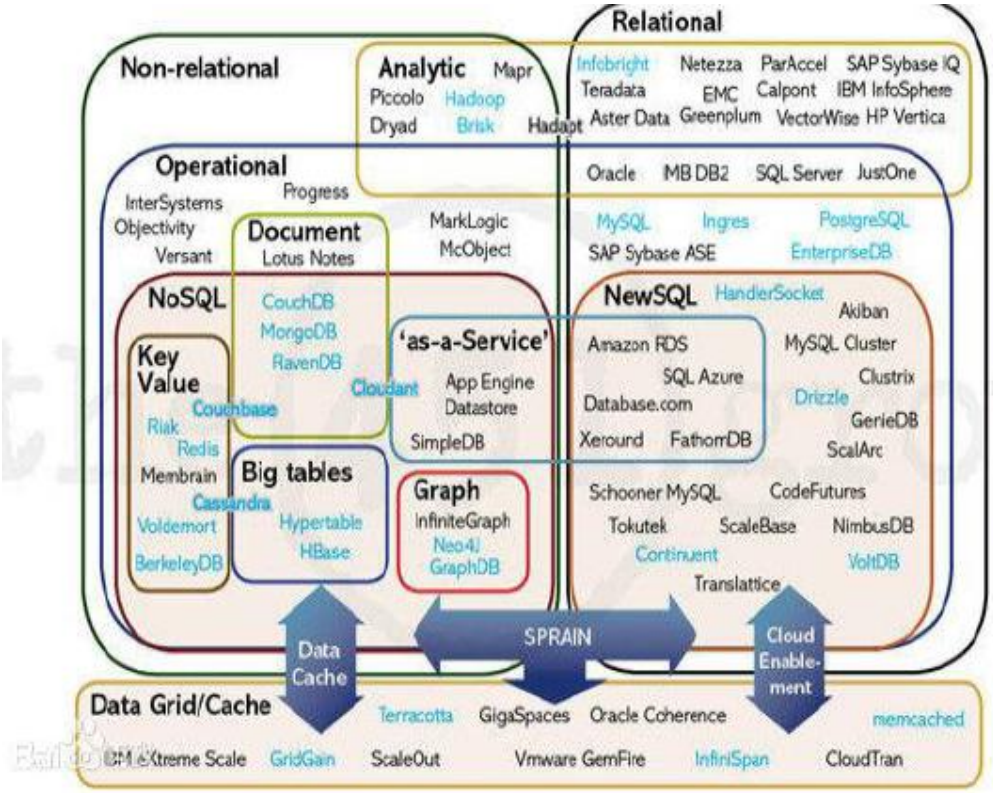
第一讲 绪论

1、数据库发展历史



2、数据库分类

- ❑ Content stores
- ❑ Document stores
- ❑ Event Stores
- ❑ Graph DBMS
- ❑ Key-value stores
- ❑ Multivalue DBMS
- ❑ Native XML DBMS
- ❑ Navigational DBMS
- ❑ Object oriented DBMS
- ❑ RDF stores
- ❑ Relational DBMS
- ❑ Search engines
- ❑ Time Series DBMS
- ❑ Wide column stores



第二讲 数据库设计概述

一、流程

（1）需求分析

（2）设计概念模型

* E-R 图、OO 模型（面向对象模型）

（3）设计逻辑模型

* 关系模型

关系模型的基本概念和基本术语共有十四个，它们分别是：

（1）关系(Relation)：一个关系对应着一个二维表，二维表就是关系名。

（2）元组(Tuple)：在二维表中的一行，称为一个元组。

（3）属性(Attribute)：在二维表中的列，称为属性。属性的个数称为关系的元或度。列的值称为属性值；

（4）（值）域(Domain)：属性值的取值范围为值域。

（5）分量：每一行对应的列的属性值，即元组中的一个属性值。 [2]

（6）关系模式：在二维表中的行定义，即对关系的描述称为关系模式。一般表示为（属性 1，属性 2，……,属性 n），如老师的关系模型可以表示为教师（教师号，姓名，性别，年龄，职称，所在系）。

（7）键(码)：如果在一个关系中存在唯一标识一个实体的一个属性或属性集称为实体的键，

即使得在该关系的任何一个关系状态中的两个元组，在该属性上的值的组合都不同。

（8）候选键(候选码)：若关系中的某一属性的值能唯一标识一个元组如果在关系的一个键中不能移去任何一个属性，否则它就不是这个关系的键，则称这个被指定的候选键为该关系的候选键或者候选码。

例如下列学生表中“学号”或“图书证号”都能唯一标识一个元组，则“学号”和“图书证号”都能唯一地标识一个元组，则“学号”和“图书证号”都可作为学生关系的候选键。

学号	姓名	性别	年龄	图书证号	所在系
S3001	张明	男	22	B20050101	外语
S3002	李静	女	21	B20050102	外语
S4001	赵丽	女	21	B20050301	管理

而在选课表中，只有属性组“学号”和“课程号”才能唯一地标识一个元组，则候选键为（学号，课程号）。

学号	课程号
S3001	C1
S3001	C2
S3002	C1
S4001	C3

（8）主键（主码）：在一个关系的若干候选键中指定一个用来唯一标识该关系的元组，则称这个被指定的候选键称为主关键字，或简称为主键、关键字、主码。每一个关系都有并且只有一主键，通常用较小的属性组合作为主键。例如学生表，选定“学号”作为数据操作的依据，则“学号”为主键。而在选课表中，主键为（学号，课程号）。

（9）主属性和非主属性：关系中包含在任何一个候选键中的属性称为主属性，不包含在任何一个候选键中的属性为非主属性。

（10）全键或者全码：一个关系模式中的所有属性的集合。

（11）外键或者外码：关系中的某个属性虽然不是这个关系的主键，或者只是主键的，但它却是另外一个关系的主键时，则称之为外键或者外码。

（12）超键或者超码：如果在关系的一个键中移去某个属性，它仍然是这个关系的键，则称这样的键为关系的超键或者超码。

（13）参照关系与被参照关系：是指以外键相互联系的两个关系，可以相互转化。

* 用户视图

视图是从一个或几个基本表(或视图)导出的表

* 各种约束

1.primary KEY:设置主键约束；

2.UNIQUE：设置唯一性约束，不能有重复值；

3.DEFAULT 默认值约束，height DOUBLE(3,2)DEFAULT 1.2 height 不输入是默认为 1,2

4.NOT NULL：设置非空约束，该字段不能为空；

5.FOREIGN key :设置外键约束。

（4）设计物理结构

（5）系统实现

（6）试运行及维护

二、概念模型的建立

1、本质：从用户的角度，对系统中数据及数据间关系进行抽象。

表现方法：

* E-R（EE-R）模型

* OO 模型

* 其它模型

抽象机制：

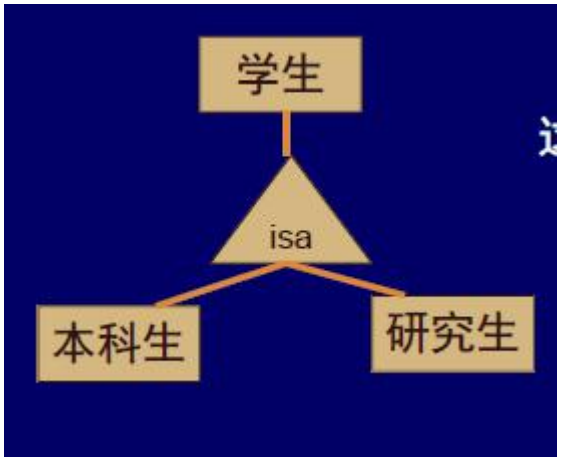
* 分类—数据

* 关联—数据间联系

* 概括/特化—数据间的蕴含关系（子类的表达）

概括：从若干个数据对象类别中抽取共同的特性，形成更高层次的数据对象类别。

特化：按照某些特性把较高层次的数据对象类别区分为若干个数据对象类别子集。

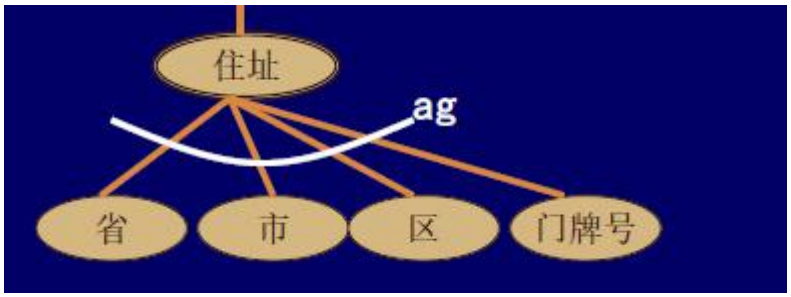


* 聚集/分解—复杂的数据关系

聚集：将若干数据对象类别或特性，关联抽象成为高一层次数据对象类别或特性

分解：将一个数据对象类别或特性，分解成为若

干个称为“部分”的数据对象类或特性。



* 隶属—弱实集

隶属：两类数据间有属于的关系，且一类数据无标识，需要靠另一类数据帮助识别。

弱实体：这种属于某一类数据又不能独立标识自己的数据类称为弱实体集。弱实体还可以来自多路联系。

三、逻辑模型的建立

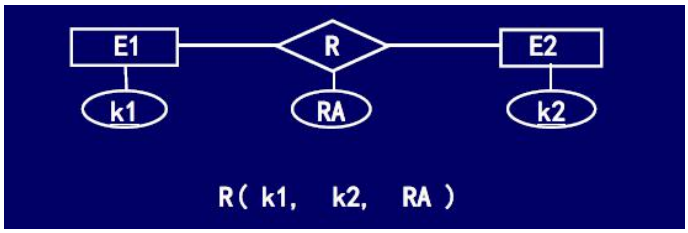
逻辑模型针对具体的数据库管理系统，目前主流关系型 DBMS

1、E-R 模型向关系模型转化

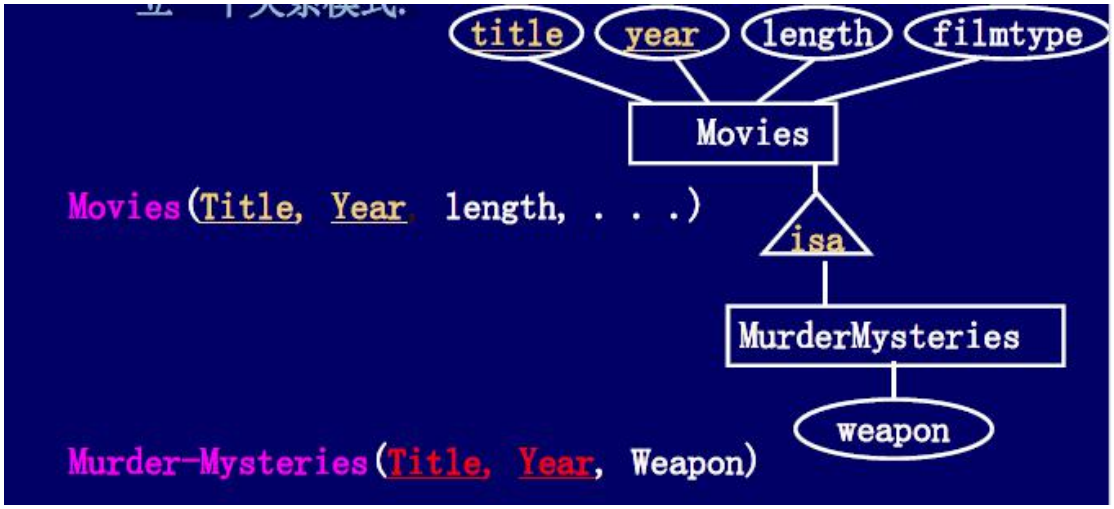
（1）变换实体集

（2）变换联系

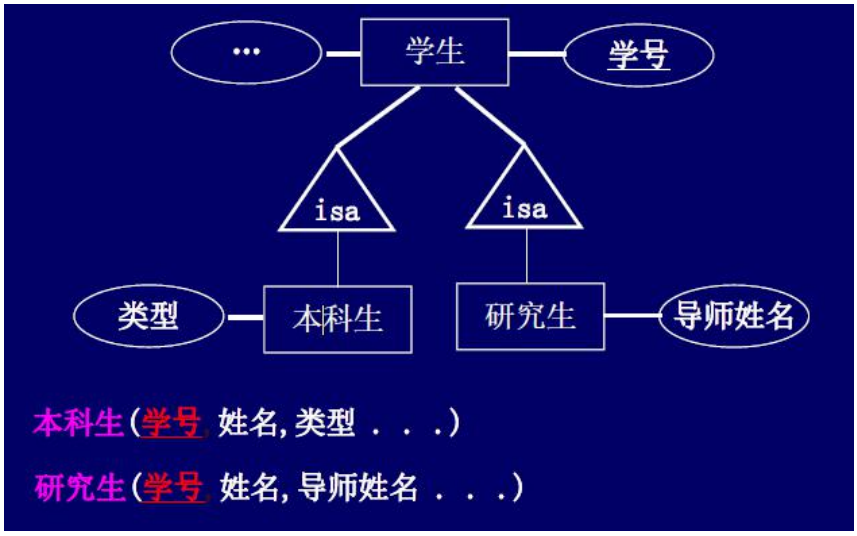
（3）子类实体集的变换



方法一：为父类实体集建立一个关系模式 为每个子类实体集分别建立一个关系模式.



方法二：如果子类实体集是不相交且全部的。



（4）弱实体集的变换：弱实体集 A 转化成一个关系模式，该关系模式的属性包括包括：A 自己的属性，以及为 A 提供主码属性的强实体集的主码。

四、设计实例

1、问题

一个旅客可以凭有效证件（如:身份证）,预订或登录班机。

旅客信息包括：身份证号，姓名，联系电话。

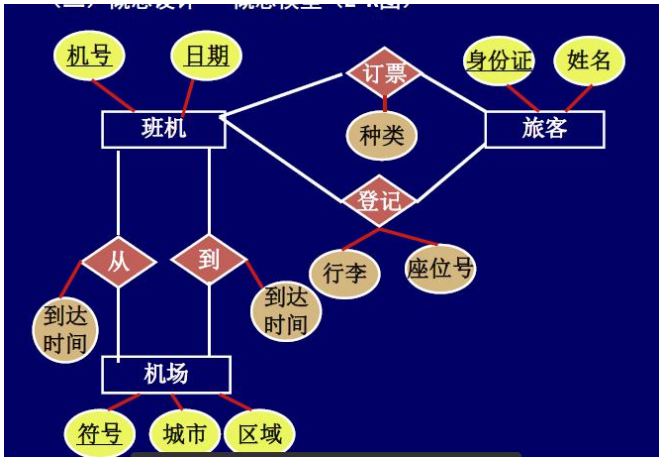
班机信息包括：班机号，（飞行）日期，可用座位，等。

机场信息包括：（机场）符号，城市，区域。

旅客订票时，需要确定：班机号，（飞行）日期，起飞机场及时间，到达机场及时间，（机票）种类（经济仓、公务仓）。

旅客登机时，需要登录信息：座位号，行李号。

2、概念模型（E-R）



2、逻辑模型

（1）关系模式设计

1）机场（符号，区域，城市）

2）班机（机号，日期，可用座位）

3）旅客（身份证号，姓名，电话）

4）从（机号，日期，符号，起飞时间）

5）到（机号，日期，符号，到达时间）

6）订票（身份证号，机号，日期，种类）

7）登记(身份证号，机号，日期，座位号，行李)

（2）规范化

“从（机号，日期，符号，起飞时间）”与

“到（机号，日期，符号，到达时间）”合并

从_到（机号，日期，起飞机场符号，到达机场符号，起飞时间，到达时间）

（3）约束设计

* 键码约束设计（实体完整性）

* 外码约束设计（参照完整性）

（4）其他数据约束设计

* 属性取值唯一性约束（UNIQUE)

* 空值约束（NOT NULL）和缺省值（DEFAULT）

* CHECK 约束和断言约束

3、视图

视图（View）是从一个或多个表（或视图）导出的表。视图与表（有时为与视图区别，也称表为基本表——Base Table）不同，视图是一个虚表，即视图所对应的数据不进行实际存储，数据库中只存储视图的定义，在对视图的数据进行操作时，系统根据视图的定义去操作与视图相关联的基本表。

4、物理设计

1. 存储设计

（1）表的存储空间估算

含：数据项（字段）、记录（元组）存储空间的估算。基础：数据类型对存储空间的需求。

含：基本表（数据文件）、索引文件、日志文件、临时表、系统表存储空间的估算。

（2）数据库的存储空间估算

（3）数据库块大小确定—块越大，性能越好?

（4）确定表数据空间初始大小及增长率。

2. 索引设计

设计依据：数据库应用

3. RDBMS 的选择

4. 日志空间和日志文件结构的选择

5. 备份数据空间设计以及备份策略的设计

5、数据库实施

1. 在选定的 RDBMS 下，创建数据库及其对象。

2. DBA 根据数据库运行情况，进行数据库的性能调优工作

第三讲 分布式数据库基本知识

3.1 什么是分布式数据库

1、分布式数据库

（1）分布式数据库是计算机网络环境中各场地（站点, Site）或节点（Node）上的数据库的**逻辑集合**。逻辑上它们属于同一系统，而物理上它们分散在用计算机网络连接的多个节点 / 场地，并统一由一个分布式数据库管理系统

（2）分布式数据库是一组数据集

（3）针对全体用户的数据库称全局数据库

（4）各节点 / 场地的数据库称局部数据库

（5）分布式数据库是虚拟的、逻辑的，只有局部数据库才是物理的数据库。

2、分布式数据库管理系统

（1）分布式数据库管理系统是分布式数据库系统中的一组软件

（2）负责管理分布环境下逻辑集成数据的存取、一致
性、有效性、完整性等
（3）可能由于各个局部数据库有不同的模型，涉及模
型转换
3、分布式数据库系统
分布式数据库系统是为地理上分散、而管理上又需要不同程度集中管理的企、事业单位提供数据管理的信息系统。
3.2 分布式数据库分类
1、是否同构：构成局部数据库的数据模型是否相同
2、性质：实现的具体技术和方法等。不同公司的产品，性质会有不同。
* 同构异质：构造相同、性质不同的局部数据库组成
* 同构同质
* 异构：构造性质等都不同
3.3 特点
（1）共享性和自治性
共享性：多个场地或节点的局部数据库在逻辑上集成为一个集体，并为分布式数据库系统的所有用户使用，这种应用称为分布式数据库的全局应用，其用户为局用户，具有共享性。
自治性：允许用户只使用本地的局部数据库，这种应用为局部应用，其用户即为局部用户，甚至局部用户所使用的数据可以不参与到全局数据库中去。这种局部用户独立于全局用户的特性即是局部数据库的自治性。
* 参与全局的局部数据 & 不参与全局数据库但又为本地共享的场地数据
（2）冗余与可用性
• 集中式数据库减少冗余
• 分布式数据库适当冗余
– 节省开销
– 提高系统可用性
– 提高自治性
冗余的不利影响
– 增加存储开销
– 增加完整性一致性控制代价
（3）事务管理的分布性
一个事务（全局事务）的执行将划分成在许多场地上执行的子事务（局部事务），子事务的执行结果合并而成全局事务的结果。
分布式事务处理的复杂性：

①由于结构性变化，要保证分布事务的操作结果具
有语义完整性和全局数据库的一致性
②与集中式数据库的事务管理在处理策略上有本质
上的差别：保证可行性和有效性，以及并行能力
③各局部子事务必须在本场地是可串行化的，同时
全局事务对系统而言也是可串行化的
④分布事务的可恢复性变得复杂
例 某银行对地处不同场地上的两个帐户转移资金
对集中式数据库，A 帐户的资金 100 元转移到 B 帐户名下，事务：T： A,A： A－100,B,B： B+100 则保证了事务的正确性。
分布式数据库：A 和 B 不在同一场地，A 和 B 都有多个副本。假设 A 在 S1、S2 各有一副本，B 在 S3、S4 各有一副本，用户请求在 S5 发出，即结果应回送至 S5，则分布事务 T：T1,T2:A,A:A-100T3,T4:B,B:B+100T5 [返回结果（或结束）消息]，此时 T 由 T1,T2,T3,T4,T5 组成。
（4）存取效率
集中式数据库：
层次模型、网络模型数据库，过程性查询，靠用户程序优化
关系模型，非过程性查询，靠查询优化处理部件进行优化
分布式数据库：
全局查询被分解成等效的子查询
优化分两级进行：全局优化和局部优化
（5）数据模型
DDB 是一个逻辑的、虚拟的数据库（被称为全局数据库（GDB））和实际分布在各场地的局部数据库（LDB）（物理的、实际存储的数据库）这样两级数据库组成。

** 系统将数据库划分为四层：全局外层（即用户层）、全局概念层、局部概念层和局部内层。
（6）数据独立性
数据独立性的基本含意是应用程序与实际的数据组织相分离，即所谓的系统透明性。
①集中式关系数据库：逻辑数据独立性和物理数据独立性
②分布式数据库：
系统透明性：用户不必关心数据模型
位置透明性：用户不必关心数据的实际存放位置

重复副本透明性：用户不必了解有多少个副本

3.4 分布式数据库问题

1、异构分布式数据库系统的问题：不同的模型和语言

2、解决策略：统一为一个公共模型和公用语言

3、模型转换两个性质：

（1）语义等价性

（2）实例的相等性

4、实现技术问题

(1)全局模式的集成（冲突解决）

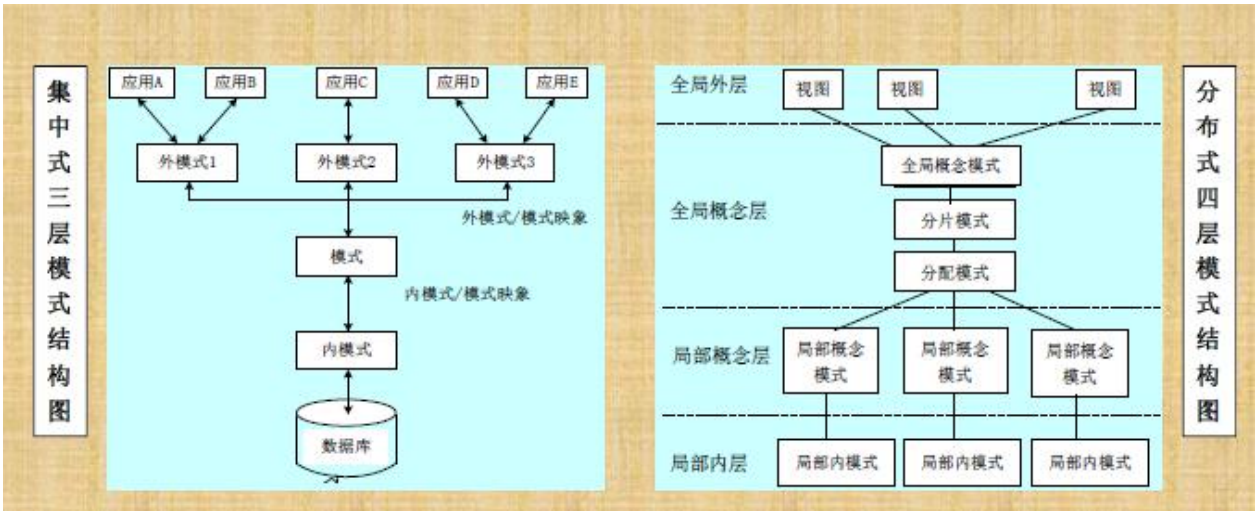
(2)查询处理问题（防止数据库性能变差）

(3)更新处理问题

** 同构分布式数据库系统的问题较少

3.5 分布式数据库系统的体系结构

1、模式结构及特点



（1）全局外层

•多个用户视图组成，是分布式数据库特定的全局用户对分布式数据库的最高层抽象。

•与集中式数据库不同，它不是从某个具体场地上的局部数据库中抽取，而是从一个虚拟的、由各局部数据库组成的逻辑集合中抽取。

•对全局用户而言，在所有分布式数据库的各个场地上，都可以认为所有的数据库都在本场地，而且他们只关心自己所使用的那部分数据。

（2）全局概念层

•分布式数据库的整体抽象，包含了全部数据特性和逻辑结构。

是对数据库的全体的描述。

•对于全局用户具有分布透明特性的分布式数据库，全局概念层

应具有三种模式描述信息，它们之间存在着映射：
①全局概念模式：该模式包含全局概念模式名、属性名、每种属性的数据类型定义和长度等。从全局概念层观察分布式数据库，它定义了全局数据的逻辑结构、逻辑分布性和物理分布性，但并不涉及全局数据在每个局部物理场地上的物理存储细节。
②分片模式：描述全局数据的逻辑划分视图。它是全局数据逻辑结构根据某种条件的划分，即成为局部的逻辑结构，每个逻辑划分即是一个片段。
③分配模式：描述局部逻辑的局部物理结构，是划分后的片段（或分片）的物理分配视图。它与集中式数据库物理存储结构的概念不同，是全局概念层的内容。

映射关系：
•全局概念模式/分片模式映射：一对多映射，即一个全局概念模式有若干个分片模式与之对应，而一个分片模式只能对应一个全局概念模式。
•分片模式/分配模式映射：可以是一对多映射，也可以是一对一映射。

** 从全局概念层观察分布式数据库，它定义了全局数据的逻辑结构、逻辑分布性和物理分布性，但并不涉及全局数据在每个局部物理场地上的物理存储细节。

（3）局部概念层

•局部概念模式描述，它是全局概念模式的子集（特殊情况下可能是全集）。
•全局概念模式经逻辑划分后，被分配在各局部场地上。在分布式数据库局部场地上，对每个全局关系有该全局关系的若干个逻辑片段的物理片段的集合，该集合是一个全局关系在某个局部场地上的物理映像，其全部则组成局部概念模式。
•如果两个场地上的所有物理映像相同，则其中一个场地上的物理映像必是另一个场地的副本，因此，两个场地的局部概念模式也必相同。

（4）局部内层
•是分布式数据库中关于物理数据库的描述，相当于集中式数据库的内层。

** 小结：分布式数据库是一组用网络联结的局部数据库的逻辑集合

（5）特点

①　将全局数据库与局部数据库分开

全局虚拟、独立于局部；局部概念层和局部内层是局部数据库；用户只需使用全局数据库操作语言

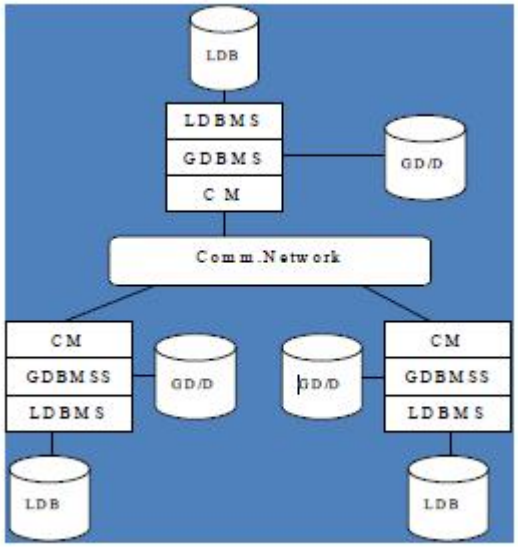
②把数据库抽象成逻辑数据库和物理数据库

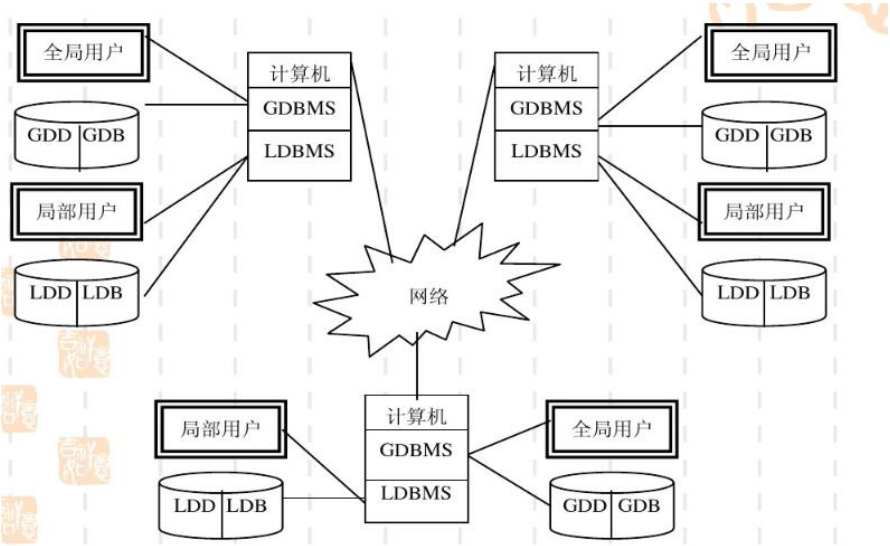
全局概念层是全局整体逻辑数据的抽象；局部概念层是局部整体逻辑数据的抽象

③把分布透明中的分片透明和分配透明相分离

2、体系结构

（1）体系结构图





计算机

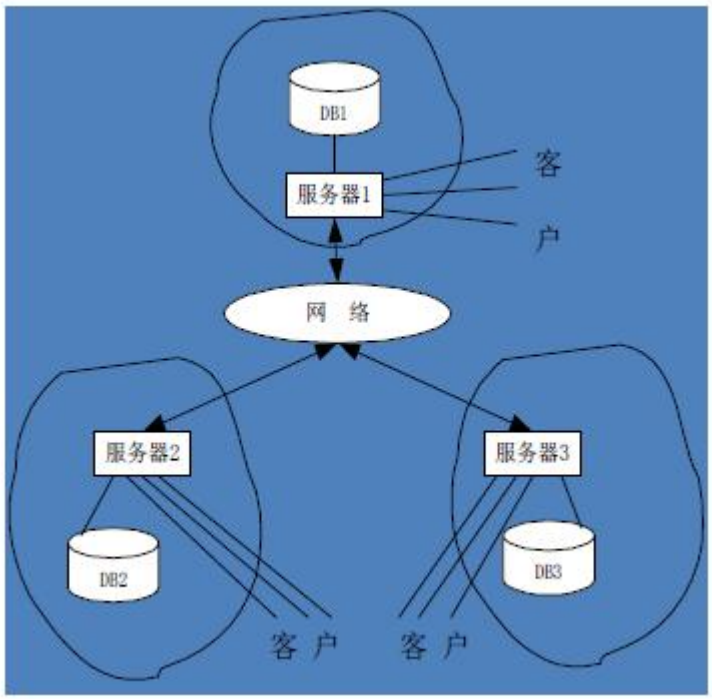
网络通讯软件

分布式数据库管理系统

分布式数据库

分布式数据库管理者

（2）系统逻辑图



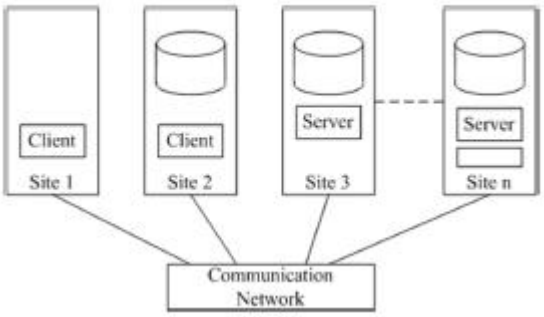
（3）分布式数据库系统环境

*分布式环境，由多个计算机设备彼此用通讯设施连接成的计算机网络环境

* 数据库系统内容渗透到网络环境中

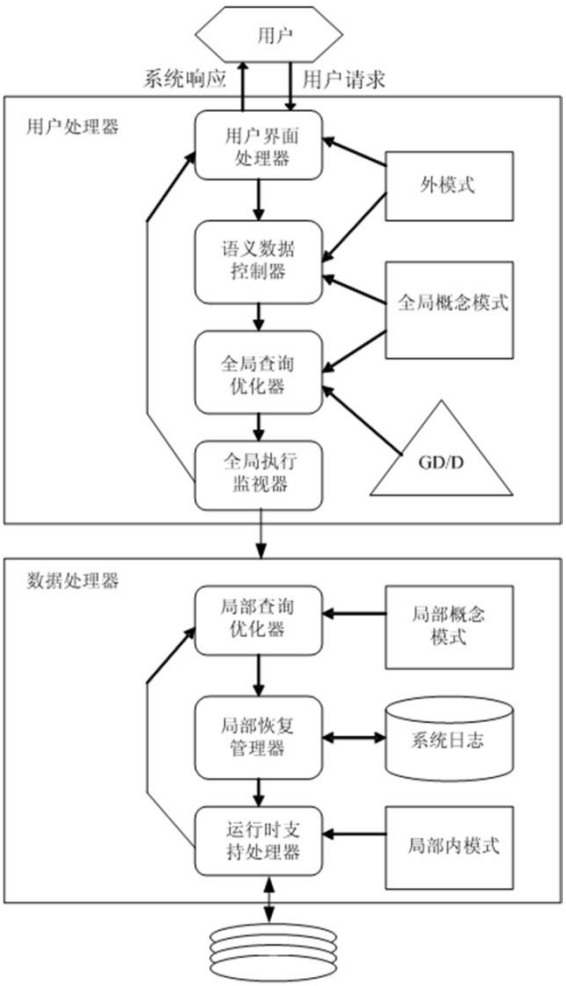
①节点/场地
②通讯设施：连接节点的物理链路和一组通讯协议
考虑：网络开销、网络延迟、网络可靠性
③网络通讯协议：ISO/OSI 标准、可自行设计

3、分布式数据库管理系统的体系结构
（1）基本功能
* 应用程序的远程数据库操作（包括查询和更新操作）
* 支持分布式数据库系统完全的或部分的透明性
* 对分布式数据库的管理和控制具有集中式数据库管理系统的功能
* 支持分布事务的并发和恢复
（2）主要成分
① 全局数据库管理系统（GDBMS）
负责管理分布式数据库（DDB）中的全局数据
② 局部数据库管理系统（LDBMS）
分布式数据库系统中各场地的数据库管理系统
③ 通讯管理程序（CM）
保证分布式数据库系统中场地间信息传送
（3）典型的分布式数据库管理系统（DDBMS）体系结构
① 客户/服务器系统结构



Server 级：场地局部数据管理
Client 级：承担分布式功能
通讯软件：信息传递
** 注意：Client 级承担 DDBS 中的 GDBMS 功能
解释：生成对多场地查询的分布执行计划；对 Server 发出命令后的管理、监督分布执行；当使用分布并发控制时，应保证全局事务的原子性；对数据的多副本应该保证副本一致性；承担有关透明性任务

② 对等分布系统结构


用户
系统响应
用户请求
用户处理器
用户界面处理器
外模式
语义数据控制器
全局概念模式
全局查询优化器
全局执行监视器
GD/D
数据处理器
局部查询优化器
局部概念模式
局部恢复管理器
系统日志
运行时支持处理器
局部内模式

第四讲 分布式数据库分片设计

一、分布式数据库设计与数据分片

1、什么是数据库设计

* 对于一个给定的应用环境，构造最优的数据库模式，建立数据库及其应用系统，使之能够有效地存储数据，满足各种用户的应用需求（信息要求和处理要求）。

2、设计的任务

•定义全局数据库的概念模式

•设计分片

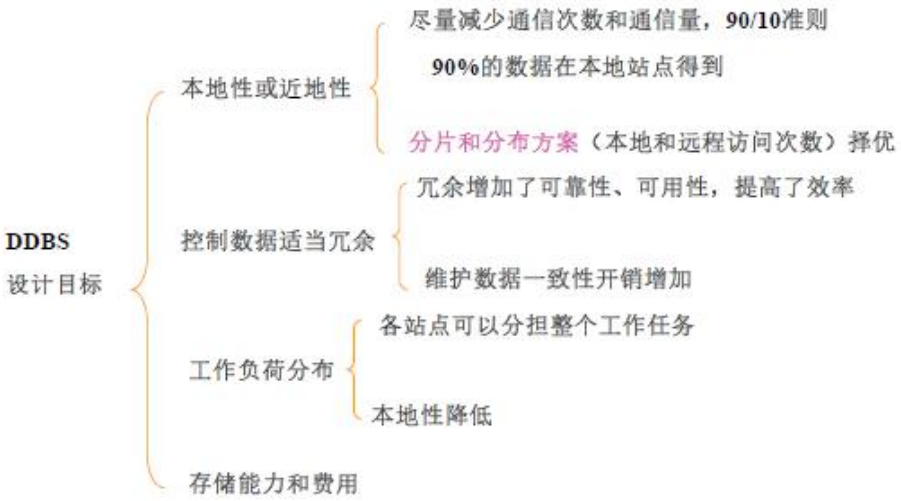
•设计片段的分配

•设计物理数据库，将概念模式映射到存储区域，并确定适当的存储方法

在分布式数据库设计过程中，必须考虑分布式数据库应用的需求，包括：应用提交的场地、应用执行的频度、每个应用所存取数据的类型、次数及统计分布等信息

3、流程

4、DDBS 设计目标



5、DDBS 设计方法

（1）自顶向下（重构法）：从头开始设计分布式数据库

（2）自底向上（组合法）：聚集现有数据库来设计分布式数据库

（3）混合方法

二、数据的分片

1、定义：对分布式数据库进行逻辑划分和实际物理分配。数据的逻辑划分称数据分片。分布式数据库中数据的存储单位，称为片段（Fragment）。对全局数据的划分，称为分片（Fragmentation），划分的结果即是片段，对片段的存储场地的指定，称为分配（Allocation）。当片段存储在一个以上场地时，称为数据复制（Replication）。如果每个片段只存储在一个场地，称为数据分割（Partition）存储。

2、作用（目的）

•减少网络传输量。如：采用数据复制，可就近访问所需信息。需频繁访问的信息分片存储在本地场地上。

•增大事务处理的局部性。局部场地上所须数据分片分配在各自的场地上，减少数据访问的时间，增强局部事务效率。

•提高数据的可用性和查询效率。就近访问数据分片或副本，可提高访问效率。同时当某一场地出故障，若存在副本，非故障场地上的数据副本均是可用的。保证了数据的可用性、数据的完整性和系统的可靠性。

•使负载均衡。减少数据访问瓶颈，提高整个系统效率

3、分片原则

假若有全局关系 R 被分片为子关系(片段)集合 R = {R1, R2, …, Rn}, 则 R 满足

– 完整性

任意$x \in R$, $\exists \mathbf{R}_i \in \mathbf{R}$ 必有 $x \in \mathbf{R}_i$， i=1,2,…,n

– 可重构性

存在函数 g 使得$R = g(R_1, R_2, \ldots, R_n)$

即，$R=\cup \mathbf{R}_i$ (水平分片），$R=\infty \mathbf{R}_i$ （垂直分片）

– 不相交性

$\mathbf{R}_i \cap \mathbf{R}_j$=空集,i≠j,i,j=1,2,…,n(水平分片)

$\mathbf{R}_i \cap \mathbf{R}_j$=主键属性,i,j=1,2,…,n(垂直分片)

4、举例

E

e#	NM	Loc	Sal
5	Joe	Sa	1000
7	Sally	Sb	2500
8	Tom	Sa	500
:			:

$\mathcal{F}$

e#	NM	Loc	Sal
5	Joe	Sa	1000
8	Tom	Sa	500
:			

站点Sa

e#	NM	Loc	Sal
7	Sally	Sb	2500
:			

站点 Sb

5、水平分片设计

（1）对全局关系执行选择操作

（2）基本水平分片（初级）

①要求

以关系自身的属性性质为基础 P = {p1, p2, …, pn}是一简单谓词集合，为保证分片的正确性，P 必须是：

- 完整的：同一分片中的任意两个元组被应用同样概率访问。
- 最小的：集合 P 中的所有谓词与应用密切相关。
- 具有完整性和最小性不是必要条件，但是对于简化分配问题有好处

②举例

EMP (E#, NAME, DEPT, JOB, SAL, TEL, …)

DEPT={1,2} JOB={‘P’, ‘-P’}

假定，应用经常查询的内容是属于部门 1 且是程序员的职员。

- 则可能有的水平分段限定
- P={ DEPT=1} （不是完整的）
- P={DEPT=1, JOB=‘P’} （是完整的、最小的）
- P={DEPT=1, JOB=‘P’, SAL>500} （完整的，不是最小的）

③如何保证分片原则

方法一：人工分析

方法二：生成具有满足分段原则的限定谓词

* 谓词生成举例

设有关系 E (e#,name,Loc,sal,A,…), 查询使用的简单谓词（Ai θ Value）是:

A<10, A>5, Loc = Sa, Loc = Sb

1、生成“小项” 谓词，选择符合要求的小项谓词

给定简单谓词集 $Pr=\{p_1, p_2, \ldots p_n\}$, 则“小项”谓词（minterm predicate）形式:

$p_1^* \wedge p_2^* \wedge \ldots \wedge p_n^*$

这里 p_k^* 是 p_k 或是 $\neg p_k$

(1) A<10 ∧ A>5 ∧ Loc=SA ∧ Loc=SB
(2) A<10 ∧ A>5 ∧ Loc=SA ∧ ¬(Loc=SB)
(3) A<10 ∧ A>5 ∧ ¬(Loc=SA) ∧ Loc=SB
(4) A<10 ∧ A>5 ∧ ¬(Loc=SA) ∧ ¬(Loc=SB)
(5) A<10 ∧ ¬(A>5) ∧ Loc=SA ∧ Loc=SB
(6) A<10 ∧ ¬(A>5) ∧ Loc=SA ∧ ¬(Loc=SB)
(7) A<10 ∧ ¬(A>5) ∧ ¬(Loc=SA) ∧ Loc=SB
(8) A<10 ∧ ¬(A>5) ∧ ¬(Loc=SA) ∧ ¬(Loc=SB)

(1) A<10 ∧ A>5 ∧ Loc=SA ∧ Loc=SB
(2) A<10 ∧ A>5 ∧ Loc=SA ∧ ¬(Loc=SB)
(3) A<10 ∧ A>5 ∧ ¬(Loc=SA) ∧ Loc=SB
(4) A<10 ∧ A>5 ∧ ¬(Loc=SA) ∧ ¬(Loc=SB)
(5) A<10 ∧ ¬(A>5) ∧ Loc=SA ∧ Loc=SB
(6) A<10 ∧ ¬(A>5) ∧ Loc=SA ∧ ¬(Loc=SB)
(7) A<10 ∧ ¬(A>5) ∧ ¬(Loc=SA) ∧ Loc=SB
(8) A<10 ∧ ¬(A>5) ∧ ¬(Loc=SA) ∧ ¬(Loc=SB)

分片结果：

R2: $5 < A < 10$ ∧ Loc=SA
R3: $5 < A < 10$ ∧ Loc=SB
R6: $A \leq 5$ ∧ Loc=SA
R7: $A \leq 5$ ∧ Loc=SB
R10: $A \geq 10$ ∧ Loc=SA
R11: $A \geq 10$ ∧ Loc=SB

2、消除无用谓词（无用段的消除依赖于应用的语义）

④ 分片数量信息

• 小项选择率（minterm selectivity） 对某一给定小项谓词用户查询可能选择到的元组数

• 访问频率（Access frequency）用户应用访问数据的频率

** 小项访问频率可以通过用户查询频率获得

⑤ 如何选择小项谓词举例

E(#, NM, LOC, SAL,…)有查询应用

Qa: select *from E where LOC=Sa Qb: select * from E where LOC=Sb

三种选择：

(1) Pr = { } R1 ={ E }

(2) Pr = {LOC=Sa, LOC=Sb}

R2={ $\sigma_{loc=Sa}$ E, $\sigma_{loc=Sb}$ E }

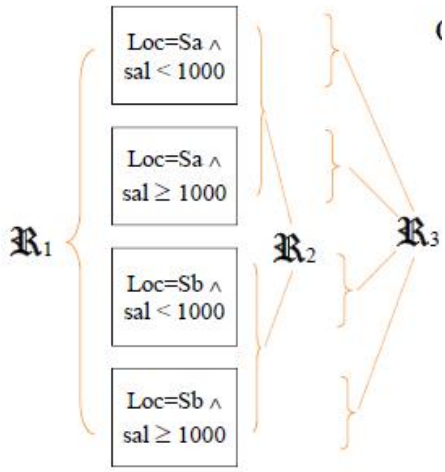
(3) Pr = {LOC=Sa, LOC=Sb, Sal<1000}

R3={ $\sigma_{loc=Sa \wedge sal<1000}$ E, $\sigma_{loc=Sa \wedge sal\geq1000}$ E,

$\sigma_{loc=Sb \wedge sal<1000}$E, $\sigma_{loc=Sb \wedge sal\geq1000}$ E }

R2 是好的，R1、R3 不好

分段情况：



理由：

对 R1：分段内元组选择概率不等

对 R3：sal 与查询应用无关

（2）导出分片：从另一个关系的属性性质或水平片段推导出来的

①例子

SC(S#, C#, GRADE)　S (S#, SNAME, AGE, SEX)

要求：将 SC 划分为男生各门课成绩和女生的各门成绩

（1）按 S 的属性导出

Define fragment SC1 as

Select SC.S#,C#,GRADE From SC, S

Where SC.S#=S.S# and SEX=‘M’

Define fragment SC2 as

Select SC.S#,C#,GRADE From SC, S

Where SC.S#=S.S# and SEX=‘F’

（2）按 S 的水平分片（SF/SM）导出

Define fragment SC1 as

Select * From SC Where S# in (Select SF.S from SF)

Define fragment SC2 as

Select * From SC Where S# in (Select SM.S from SM)

2、垂直分片设计

（1）垂直分片和垂直集群

通过“投影”操作把一个全局关系的属性分成

若干组，基本目标是将使用频繁的属性聚集在

一起

• 全局关系 R={Ri},i=1,2,…,n

垂直分片：如果属性 A∈R,必有 A∈Ri,i=1,2,…,n,而且

Ri∩Rj=Ap,i≠j，Ap 为 R 的码或元组标识符，则称

{Ri},i=1,2,…,n}是关系 R 的一个垂直分片。

垂直群集：如果属性 A∈R,必有 A∈Ri,i=1,2,…,n,而且 Ri∩Rj=

（Ap, A-p),i≠j，A-p 为 R 的一个或多个非码属性时，

称{Ri},i=1,2,…,n}是关系 R 的一个垂直群集。

（2）举例

EMP(E#, NAME, SAL, TEL, MAGNUM, DEPT)

- 假定 Key: E# 主要应用: Sa 站点查询 NAME,SAL, TEL; Sb 站点查询 NAME, MAGNUM, DEPT
- 垂直分片:EMP1(E#, NAME, SAL, TEL) EMP2(E#, MAGNUM, DEPT)
- 垂直群集:EMP1(E#, NAME, SAL, TEL) EMP2(E#, NAME, MAGNUM, DEPT)

垂直分片例子

E

#	NM	Loc	Sal
5	Joe	Sa	1000
7	Sally	Sb	2500
8	Fred	Sa	1500
…			

E1

#	NM	Loc
5	Joe	Sa
7	Sally	Sb
8	Fred	Sa
…		

E2

#	Sal
5	1000
7	2500
8	1500
…	

（3）属性的亲和关系

- 非键属性 $A_1, A_2, \ldots, A_n$
- 应用 $Q_1, Q_2, \ldots, Q_m$
- $freq(Q_i) = Q_i$ 的访问频率

$$aff(A_i, A_j) = \sum_{k|use(Q_k, A_i) \wedge use(Q_k, A_j)} freq(Q_k)$$

（4）属性亲和矩阵

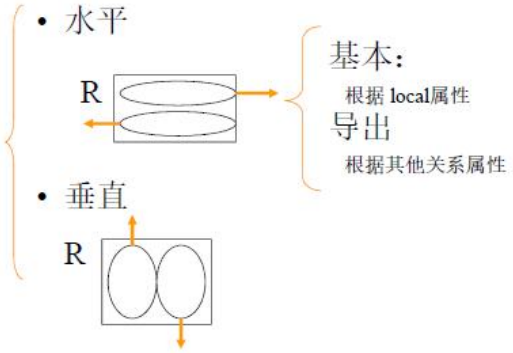
	A1	A2	A3	A4	A5
A1	96	50	45	1	0
A2	50	100	48	2	0
A3	45	48	97	0	4
A4	1	2	0	78	75
A5	0	0	4	75	79

$R_1[K, A_1, A_2, A_3]$ $R_2[K, A_4, A_5]$

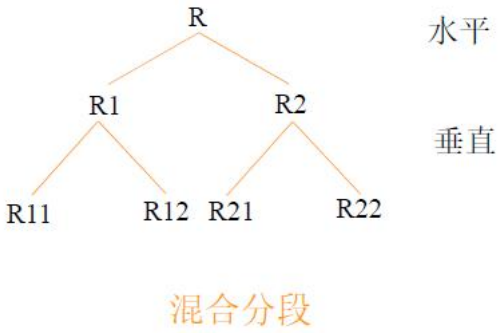
（5）垂直分片算法

- 穷举属性亲和矩阵的列排列
  - 行与列要同时调整
- 发现好的“分割点”
  - 极大化每个分割内的亲合力（affinity），极小化跨分割的访问

（6）数据的分片设计



（7）数据的混合分片



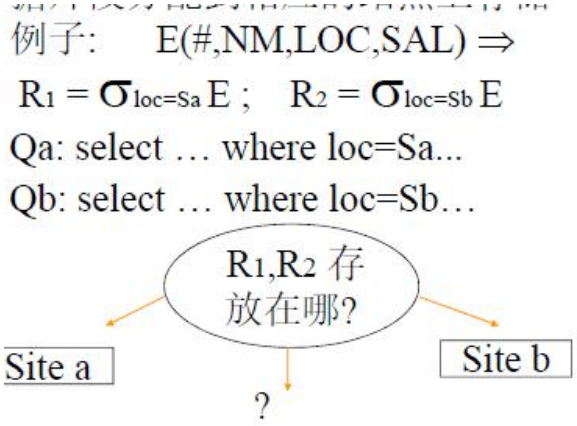
混合分段

四、数据分配

1、定义

在满足用户需求的前提下，把设计好的数据片段分配到相应的站点上存储。

2、举例



3、优化问题

（1）段的最好配置（最好的冗余副本数）

① 极小化查询响应时间

② 极大化吞吐量

③ 极小化代价

…

（2）约束

① 有效的存储空间

② 有效的带宽、站点处理能力

③ 保持 90%的响应时间低于 x

4、分配的简化模型

单个片段 F　　站点 $S_1, \ldots S_m$

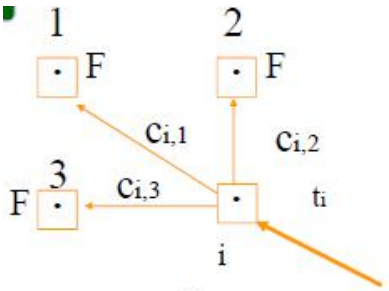
变量　$X_1, \ldots, X_m$

$$Xj = \begin{cases} 0 & \text{如果 F 不在 } S_j \text{ 上存储} \\ 1 & \text{如果 F 在 } S_j \text{ 上存储} \end{cases}$$

Total cost = Read Cost + Write Cost +Storage Cost

确定 Xj 的值, 1<=j<=m, 使总代价极小

① 读代价



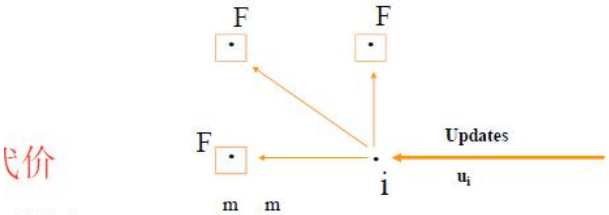
$$\text{Read cost} = \sum_{i=1}^{m} [ti \times \underset{j}{\text{MIN}}\ Cij]$$

i:　读申请源站点

ti:　站点Si上的读申请激活次数

Cij:　从 Si读Sj站点分段F的代价

② 写代价



$$\text{Write cost} = \sum_{i=1}^{m} \sum_{j=1}^{m} Xj\ ui\ C'ij$$

i:　写申请源站点

j:　被更新站点

Xj:　0　if F not stored at S_j

　　1　if F stored at S_j

ui:　站点 S_i 上更新激活次数

C'ij:　从站点 S_i 更新 S_j 分段 F 的代价

③ 存储代价

$$\text{Store Cost} = \sum_{i=1}^{m} X_i\, d_i$$

X_i: $\begin{cases} 0 & \text{if F not stored at } S_i \\ 1 & \text{if F stored at } S_i \end{cases}$

d_i: 站点 S_i 存储分段 F 的代价

④ 目标函数

$$\min\left\{\sum_{i=1}^{m}\left[t_i \times \underset{j}{\text{MIN}}\, C_{ij} + \sum_{i=1}^{m}\sum_{j=1}^{m} X_j \times u_i \times C'_{ij}\right] + \sum_{i=1}^{m} X_i \times d_i\right\}$$

即使最简单的公式也是 NP-完全问题。

** 通常方法：将片段分配在被局部访问位置

5、分配方法

（1）“最佳适应”方法（非冗余分配）

$$B_{ij} = \Sigma_k F_{kj} \times N_{ki}$$

（2）“所有得益站点”方法（冗余分配）

$$B_{ij} = \Sigma_k F_{kj} \times R_{ki} - c \times \Sigma_k \Sigma_{j' \neq j} F_{kj'} \times U_{ki}$$

i 片段下标　　j 站点下标

k 应用下标　　F_{kj} 应用k 在站点j上激活的频率

R_{ki} 应用k被激活一次,对片段i读的次数

U_{ki} 应用k被激活一次,对片段i写的次数

N_{ki} 应用k被激活一次,对片段i读写的总次数

B_{ij} 片段i放在站点j得益

6、水平分片情况

（1）最佳适应法

将片断 Ri 分配到访问 Ri 次数最多的那个站点上：$B_{ij}= \Sigma_k F_{kj} * N_{ki}$

（2）所有得益站点法

* 将片断 Ri 的副本分配到所有得益站点 j 上：$B_{ij}= \Sigma_k F_{kj}*R_{ki} -c*\Sigma_k \Sigma_{j'\neq j} F_{kj'}*U_{ki}$

如 Bij ＞0，则站点 j 是得益站点，放置 Ri 的一个副本

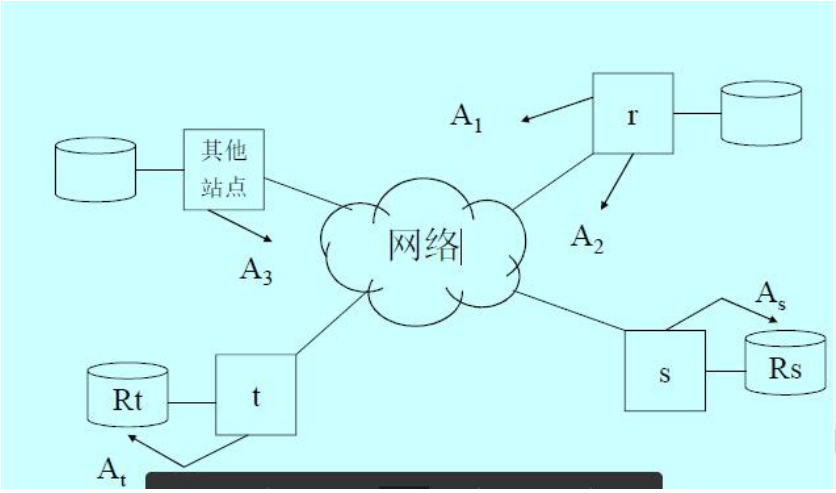
** 得益：读得益-附加产生更新花费

7、垂直分片情况

假设关系 R 垂直分片 Rt 和 Rs, Rs 分配到 s 站点, Rt 分配到 t 站点。

- 应用组As: 自站点s发出, 只使用Rs, 得益
  $BAs = \Sigma_k F_{ks} \times N_{ks} (k \in As)$
- 应用组Ar: 自站点t发出, 只使用Rt, 得益
  $BAt = \Sigma_k F_{kt} \times N_{kt} (k \in At)$
- 应用组A1: 由站点r发出, 只使用Rs,要远程，损失
  $BA1 = \Sigma_k F_{kr} \times N_{ks} (k \in A1)$
- 应用组A2: 由站点r发出, 只使用Rt, 要远程，损失
  $BA2 = \Sigma_k F_{kr} \times N_{kt} (k \in A2)$
- 应用组A3: 由不同于站点r,s,t的站点发出, 要访问Rt和Rs, 损失
  $BA13 = \Sigma_{\{s,t\}} \Sigma_k F_{kj} \times N_{ki} (k \in A3, j \neq r,s,t)$

总的分配得益：$B_{ist} = BAs + BAt - BA1 - BA2 - BA3$



五、小结

1、练习 1

已知有如下两种段分配:

A> R1 在 Site1, R2 在 Site2, R3 在 Site3.

B> R1 和 R2 在 Site1, R2 和 R3 在 Site3.

另已知有如下应用(所有应用的频率相同)

A1: 在 Site1 上发出，读 5 个 R1 记录, 5 个 R2 记录

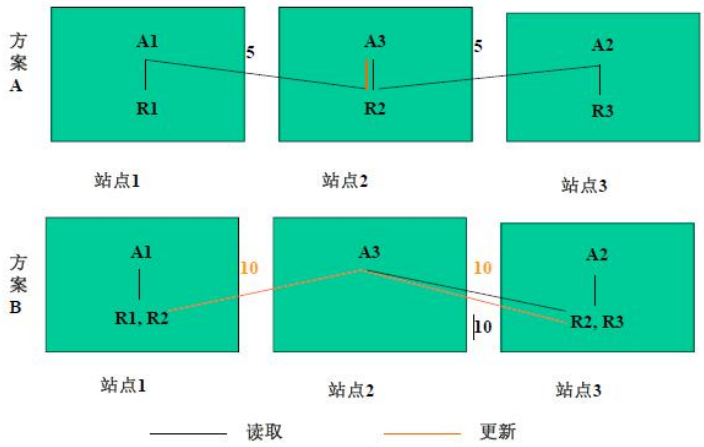
A2: 在 Site3 上发出，读 5 个 R3 记录, 5 个 R2 记录

A3: 在 Site2 上发出，读 10 个 R2 记录.

问:

1. 如果以局部访问为主要设计目标，那个分配较优?

2. 假定 A3 改为要修改 10 个 R2 记录，并仍以局部访问为其设计目标，则那个分配方案较优?



- 解：首先完善模型如下：

假定：B方案中Site1上的R2和Site3上的R2是完全相同的，即冗余；本地读写的代价可忽略不计；
所有异地读操作的单位记录代价均相等，记为1；
所有异地写操作的单位记录代价均相等，记为1。

	1 A方案	B方案	2 A方案	B方案
A1代价	5	0	5	0
A2代价	5	0	5	0
A3代价	0	10	0	10+10+10

故可知，第1个问题，A、B两种分配方案代价相等。
第2个问题，A分配方案较优。

2、数据分布模式定义

DDB 的数据分布在四层模式中属全局概念层的描述，它是分布式数据库的整体抽象，也是分布式数据库与集中式数据库抽象的最特殊的一点。这里讨论描述全局概念层中关于数据分布的模式定义语句，而四层模式中其它三层的模式定义与集中式数据库类似。

在全局概念层中，有全局概念模式、分片模式、分配模式三种模式。所以，对全局层模式定义语句应包括这三方面的描述，语句的词法和语法视设计的风格而

定。

第五讲 分布式数据库查询处理与优化

一、分布式查询优化概述

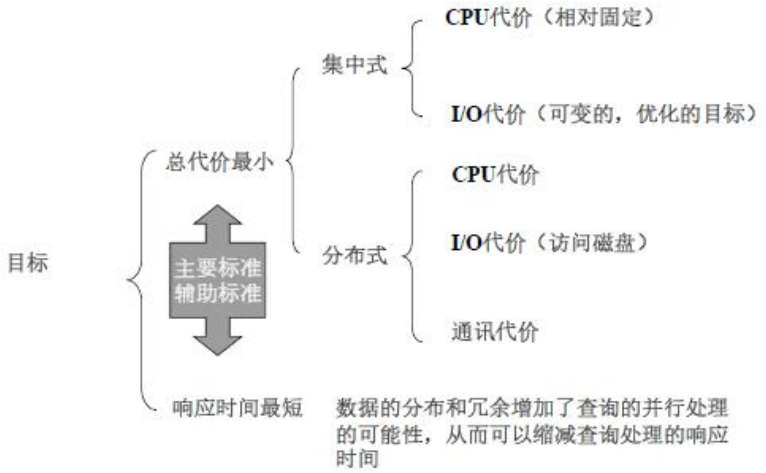
1、目标

（1）查询时需要处理的问题

集中式：查询转换为代数表达式；从所有等价表达式中选择最优的代数表达式

分布式：集中式的问题；站点之间交换数据的操作；选择最优的执行站点；数据被传送的方式

（2）目标



2、优化准则和代价分析

（1）准则

使得通讯费用最低和响应时间最短，即以最小的总代价，在最短的响应时间内获得需要的数据。

1. 通讯费用与所传输的数据量和通信次数有关

2. 响应时间和通信时间有关，也与局部处理时间有关

（2）查询代价分析

1. 远程通讯网络

局部处理时间可以忽略不计，减少通讯代价是主要目标

2. 高速局域网

传输时间比局部处理时间要短很多，以响应时间作为优化目标，局部处理时间是关键

3、分布式查询策略的重要性

* 例子

S(s#, sname, age, sex) 10^4 元组 Site A

C(c#, cname, teacher) 10^5 元组 Site B

SC(s#, c#, grade) 10^6 元组 Site A

每个元组长度 100bit, 通讯传输速度 10^4 bit/sec,通讯延迟 1sec

查询: 所有选修 maths 课的男生学号和姓名.

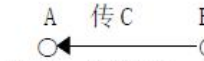
```
SELECT   s#, sname
FROM   S, C, SC
WHERE  S.s#=SC.s#  AND
          C.c#=SC.c#  AND
          sex='男'   AND cname='maths';
```

** 代价公式：QC=I/O 代价+CPU 代价+通讯代价

** 通讯代价：TC=传输延迟时间 C0+（传输数据量 X/数据传输速率 CI）

四种查询策略

策略 1:

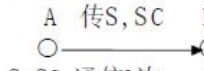


A 传 C B 把关系C传输到A地，在A地处理查询

S,SC 通信1次 C $T1 = 1 + (10^5 * 100 / 10^4)$

$\approx 10^3$ 秒 $\approx$ 16.7分钟

策略 2:



A 传S, SC B 把关系S和SC传输到B地，在B地处理查询

S,SC 通信2次 C $T2 = 2 + (10^4+10^6) * 100 / 10^4$

$\approx$ 10100秒 $\approx$ 2.8小时

策略 3：

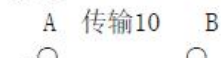
A 传输10^5 B 先在A地求出男生选课元组，有10^5 个

S,SC 通信1次 C 再把结果传输到B地，在B地执行查询，

$T5 = 1 + (10^5 * 100) / 10^4$

$\approx$ 1000 秒 $\approx$ 16.7 分

策略 4：



A 传输10 B 先在B地求出为‘MATHS’的元组，有10个

S,SC 通信 1次 C 再把结果传输到 A 地，在A 地执行查询，

$T6 = 1 + (10 * 100) / 10^4$ $\approx$ 1秒

二、分布式查询优化的基础知识

1、关系代数知识

* 关系代数：是根据查询来生成新表的方法的集合。

（1）关系代数的运算类型

① 集合运算：表实际上是“行”的集合，集合的运算是只涉及“行”的运算；

名称	符号	示例
并	∪	R∪S，或R UNION S
交	∩	R∩S，或R INTERSECT S
差	–	R–S，或R MINUS S
笛卡儿积	×	R×S，或R TIMES S

② 专门的关系运算：即涉及行，又涉及列

名称	符号	示例
投影	π	$\pi_{Ai1\ldots Aik}R$
选择	σ	$\sigma_{C\#='C2'}R$
连接	⋈	R⋈S，或R JOIN S
除	÷	R÷S，或R DIVIDEBY S

③ 其他关系运算

名称	符号	键盘格式	示例
外连接	$\bowtie_O$	OUTERJ	R$\bowtie_O$S，或R OUTERJ S
左外连接	$\bowtie_{LO}$	LOUTERJ	R$\bowtie_{LO}$S，或R LOUTERJ S
右外连接	$\bowtie_{RO}$	ROUTERJ	R$\bowtie_{RO}$S，或R ROUTERJ S
θ连接	$\bowtie_{Ai\,\theta\,Bj}$	JN(Ai θ Bj)	R$\bowtie_{Ai\,\theta\,Bj}$S，或R JN(Ai θ Bj) S

这里，θ是比较运算符，可以是>, <, >=, <=, =, <>

（2）交、并、差运算

R1

A	B	C
a1	b1	c1
a1	b2	c2
a2	b2	c1

R2

A	B	C
a1	b2	c2
a1	b3	c2
a2	b2	c1

R1∪R2

A	B	C
a1	b1	c1
a1	b2	c2
a1	b3	c2
a2	b2	c1

（3）广义笛卡尔积

广义笛卡尔积

两个分别为n目和m目的关系R和S的广义笛卡尔积是一个(n+m)列的元组的集合。元组的前n列是关系R的一个元组，后m列是关系S的一个元组。若R有k1个元组，S有k2个元组，则关系R和关系S的广义笛卡尔积有k1×k2个元组。记作：

$$R\times S=\{\widehat{t_r\ t_s}\mid t_r\in R\ \wedge\ t_s\in S\}$$

R1

A	B	C
a1	b1	c1
a1	b2	c2
a2	b2	c1

R2

A	B	C
a1	b2	c2
a1	b3	c2
a2	b2	c1

R1×R2

A	B	C	A	B	C
a1	b1	c1	a1	b2	c2
a1	b1	c1	a1	b3	c2
a1	b1	c1	a2	b2	c1
a1	b2	c2	a1	b2	c2
a1	b2	c2	a1	b3	c2
..	..	..	..	..	..

（4）连接运算（θ连接）

连接运算是从两个关系的笛卡尔积中选取属性间满足一定条件的元组。

记做：$R\underset{F}{\infty}S$. 其中，F是条件表达式，它涉及到对两个关系中的属性的比较。

$$R\underset{A\,\theta\,B}{\bowtie}S=(\widehat{t_r\ t_s}\mid t_r\in R\ \wedge\ t_s\in S\ \wedge\ t_r[A]\ \theta\ t_s[B])$$

例　设关系R、S如下图：　$R\underset{C<E}{\infty}S$

R

A	B	C
a1	b1	5
a1	b2	6
a2	b3	8
a2	b4	12

S

B	E
b1	3
b2	7
b3	10
b3	2
b5	2

$R\underset{C<E}{\infty}S$

A	R.B	C	S.B	E
a1	b1	5	b2	7
a1	b1	5	b3	10
a1	b2	6	b2	7
a1	b2	6	b3	10
a2	b3	8	b3	10

（5）连接运算（自然连接）

自然连接

另一种是自然连接。自然连接是一种特殊的等值连接，它要求两个关系中进行比较的分量必须是相同的属性组，并且要在结果中把重复的属性去掉。

$$R\bowtie S=(\widehat{t_r\ t_s[\overline{B}]}\mid t_r\in R\ \wedge\ t_s\in S\ \wedge\ t_r[B]=t_s[B])$$

例6　关系R、S的自然连结：$R\infty S$

$R\infty S$

A	B	C	E
a1	b1	5	3
a1	b2	6	7
a2	b3	8	10
a2	b3	8	2

$R\underset{R.B=S.B}{\infty}S$

A	R.B	C	S.B	E
a1	b1	5	b1	3
a1	b2	6	b2	7
a2	b3	8	b3	10
a2	b3	8	b3	2

（6）半连接

在R、S自然连接后仅保留对R的属性的投影，记为：R ∝ S

例　关系R、S的半连接：

R

A	B	C
a1	b1	5
a1	b2	6
a2	b3	8
a2	b4	12

S

B	E
b1	3
b2	7
b3	10
b3	2
b5	2

R∞ S

A	B	C	E
a1	b1	5	3
a1	b2	6	7
a2	b3	8	10
a2	b3	8	2

R ∝ S

A	B	C	
a1	b1	5	
a1	b2	6	
a2	b3	8	

（7）关系代数表达式

设教学数据库中有三个关系：

学生关系S（S#，SNAME，SD，AGE）

课程关系C（C#，CNAME, TEACHER）

学习关系SC（S#，C#，GRADE）

例　检索学习课程号为C2的学生学号与成绩

$\sigma_{C\#='C2'}(SC)$

学号 S#	课程号 C#	学习成绩 GRADE
S1	C2	A
S2	C2	C
S3	C2	B
..	..	..

$$\pi_{S\#,GRADE}(\sigma_{C\#='C2'}(SC))$$

SC

学号 S#	课程号 C#	学习成绩 GRADE
S1	C1	A
S1	C2	A
S1	C3	A
S1	C5	B
S2	C1	B
S2	C2	C
S2	C4	C
S3	C2	B
..	..	..

例 检索学习课程号为C2的学生学号和姓名

S

学号 S#	学生姓名 SNAME	所属系名 SD	学生年龄 SA
S1	A	CS	20
S2	B	CS	21
S3	C	MA	19
S4	D	CI	19
S5	E	MA	20
..	..	..	..

SC

学号 S#	课程号 C#	学习成绩 GRADE
S1	C1	A
S1	C2	A
S1	C3	A
S1	C5	B
S2	C1	B
S2	C2	C
..	..	..

S ∞ SC

学号 S#	学生姓名 SNAME	所属系名 SD	学生年龄 SA	课程号 C#	学习成绩 GRADE
S1	A	CS	20	C1	A
S1	A	CS	20	C2	A
S1	A	CS	20	C3	A
S1	A	CS	20	C5	A
S2	B	CS	21	C1	B
S2	B	CS	21	C2	C
..	..	..	..	..	..

$\sigma_{C\#='C2'}(S \infty SC)$

S#	SNAME
S1	A
S2	B

$\pi_{S\#,SNAME}(\sigma_{C\#='C2'}(S \infty SC)) = \pi_{S\#,SNAME}(S \infty_{C\#='C2'} SC)$

例 检索学习课程号为C2或C3的学生学号和所在系

S

学号 S#	学生姓名 SNAME	所属系名 SD	学生年龄 SA
S1	A	CS	20
S2	B	CS	21
S3	C	MA	19
S4	D	CI	19
S5	E	MA	20
..	..	..	..

SC

学号 S#	课程号 C#	学习成绩 GRADE
S1	C1	A
S1	C2	A
S1	C3	A
S1	C5	B
S2	C1	B
S2	C2	C
..	..	..

$\pi_{S\#,SD}(S \infty \pi_{S\#}(\sigma_{C\#='C2' \vee C\#='C3'}(SC)))$

2、用关系代数和 SQL 语句表示一个查询

关系代数基本操作：

并（∪）、交（∩）、笛卡尔积（ × ）、选择（ ）、投影（ ）

关系代数导出操作：

差（—）、除（÷）、θ 连接（ ∞ θ ）、自然连接（∞）、半连接（∝）

* SQL 与代数的等价描述

例一：SELECT sname FROM S, SC WHERE S.s#=SC.s# and SC.c#=‘c03’;

代数描述： $\pi_{sname}(\sigma_{s.s\#=SC.s\# \text{ and } SC.c\#='c03'}(S \times SC))$

例二： SELECT sname FROM S WHERE S.s# in (SELECT SC.s# FROM SC WHER c#=‘c03’);

代数描述： $\pi_{sname}(\sigma_{s.s\#=SC.s\#}(S \times \sigma_{SC.c\#='c03'} SC))$

例三：SELECT sname FROM S , (SELECT SC.s# FROM SC WHER c#=‘c03’) SCC WHERE S.s# = SCC.s# ;

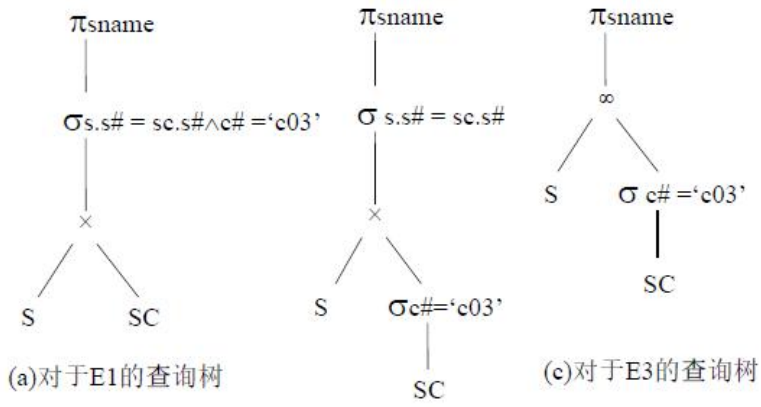
代数描述：$\pi_{sname}(S \infty \sigma_{SC.c\#='c03'} SC)$

3、查询树

（1）叶子：表示已知关系

（2）节点：表示一个一元或二元操作符

（3）树根：查询结果



(a)对于E1的查询树　(b)对于E2的查询树　(c)对于E3的查询树

4、等价变换规则的概念和术语

（1）一元操作：只涉及一个操作对象　σ (SL), π (PJ)

（2）二元操作：涉及两个操作对象

∪ (并), ∩（交）, − (差), × (笛卡尔积),
∞θ（θ连接）, ∞ (自然连接), ∝ (半连接), ÷
（除）

（3）空集的等价变换规则

$R \cup \emptyset = R \quad R \cap \emptyset = \emptyset \quad R \infty \emptyset = \emptyset \quad \emptyset \propto R = \emptyset$

$\emptyset - R = \emptyset \quad R - \emptyset = R \quad R \infty\theta \emptyset = \emptyset \quad R \times \emptyset = \emptyset$

$\sigma(\emptyset) = \emptyset \quad \pi(\emptyset) = \emptyset$

（4）自身操作的等价

$R \cup R = R \quad R \cap R = R \quad R \infty R = R$

（5）一元操作的等价

$\sigma_{F1}(\sigma_{F2}(R)) = \sigma_{F1 \text{and } F2}(R)$

F只涉及$A_1 \ldots A_n$的属性：

$$\pi_{A1,\ldots,An}(\sigma_F(E)) = \sigma_F(\pi_{A1,\ldots,An}(E))$$

F还涉及不在$A_1 \ldots A_n$中的$B_1 \ldots B_m$的属性：

$$\pi_{A1,\ldots,An}(\sigma_F(E)) = \pi_{A1,\ldots,An}(\sigma_F(\pi_{A1,\ldots,An,B1,\ldots,Bm}(E)))$$

（6）交换律

$O_1(O_2(R)) = O_2(O_1(R))$

条件:

- O_1 O_2 是 选择操作 时总成立
- O_1 O_2 是 投影操作 时要求其属性集合相等
- O_1 与 O_2 是投影和选择操作时:

$\pi_{A1,\ldots An}(\sigma_F(R)) = \sigma_F(\pi_{A1,\ldots An}(R))$ 的条件是 F中的属性是 A1, …. An 的子集

$R \infty S = S \infty R \quad R \times S = S \times R$

$R \cup S = S \cup R \quad R \cap S = S \cap R$

$R \propto S \neq S \propto R \quad R - S \neq S - R$

（7）结合律

R B (S B T) = (R B S) B T

B: 二元操作，∞，×，∪，等总成立

（8）分配率

$O(R \ B \ S) = O(R) \ B \ O(S)$

O:一元操作 B:二元操作

- $\sigma_F(R \times S)$ 其中F = F1 and F2,
  若F1有R属性，F2有S属性，则

$\sigma_F(R \times S) = \sigma_{F1}(R) \times \sigma_{F2}(S)$

若F1只有R属性，F2有R与S属性，则

$\sigma_F(R \times S) = \sigma_{F2}(\sigma_{F1}(R) \times S)$

$\sigma_F(R \cup S) = \sigma_F(R) \cup \sigma_F(S)$

$\sigma_F(R - S) = \sigma_F(R) - \sigma_F(S)$

$\sigma_F(R \infty S) = \sigma_F(R) \infty \sigma_F(S)$

三、分布式查询的分类与结构层次

1、分布式查询分类

（1）局部查询：只涉及本地单个站点的数据查询，优化同集中式

- 选择和投影早做，中间结果大大减少
- 连接前进行预处理（属性排序、属性索引）
- 同时执行一串投影和选择操作

• 找出公共子表达式

（2）远程查询

只涉及单个站点的数据，但要远程查询，选择站点

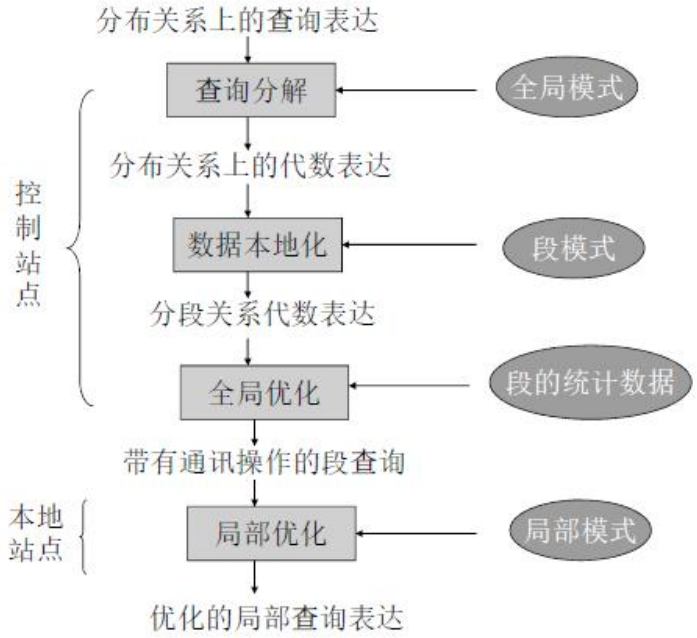
* 选择查询应用最近的冗余分配站点

（3）全局查询

涉及多个站点数据，优化复杂，基本方法如下：

- 具体化
  - 对查询进行分解，确定查询使用的物理副本，落实查询对象
  - 非冗余具体化，所有要访问对象只有一个副本
  - 冗余具体化，多个副本，研究如何如何选择副本，使通信代价最小
- 确定操作执行的顺序
  - 确定二元操作连接和并操作的顺序
    - 先执行所有连接操作，再执行并操作
    - 先执行部分并操作，再执行连接操作
  - 选择和投影尽可能早进行
- 确定操作执行的方法
  - 把若干个操作连接起来在一次数据库访问中完成，确定可用的访问路径
  - 连接方法在查询优化中起着重要作用
- 确定执行的站点
  - 执行站点不一定是发出查询的站点
  - 考虑通讯费用和执行效率

2、分布查询的层次



- 查询分解
  - 将查询问题（SQL）转换成一个定义在全局关系上的关系代数表达式
  - 需要从全局概念模式中获得转换所需要的信息
- 数据本地化
  - 具体化全局关系上的查询，落实到合适的片段上的查询

- 即将全局关系上的关系代数表达式变换为相应片段上的关系代数表达式

• 全局优化
- 优化目标是寻找一个近于最优的执行策略（操作次序）
- 输出是一个优化的、片段上的关系代数查询

• 局部优化
- 输入是局部模式
- 它由该站点上的 DBMS 进行优化

四、基于关系代数等价变换的查询优化处理

1、基本原理和实现方法

（1）基本原理

1. 查询问题——〉关系代数表达式
2. 分析得到查询树
3. 进行全局到片段的变换得到基于片段的查询树
4. 利用关系代数等价变换规则的优化算法，尽可能先执行选择和投影操作

（2）优化算法

1. 连接和合并尽可能上提（树根方向）
2. 选择和投影操作尽可能下移（叶子方向）

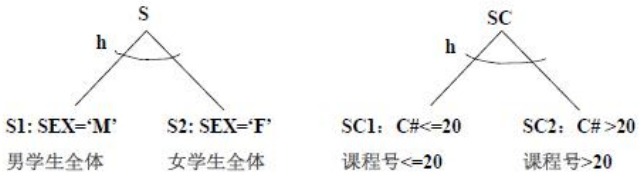
（3）实现步骤和方法
- 转换一：查询问题——〉关系代数表达式
- 转换二：关系代数表达式——〉查询树
- 转换三：全局查询树分拆成片段查询树
- 优化：利用关系代数等价变换规则的优化算法，优化查询树，进而优化查询

2、查询树优化处理举例

（1）水平分片举例

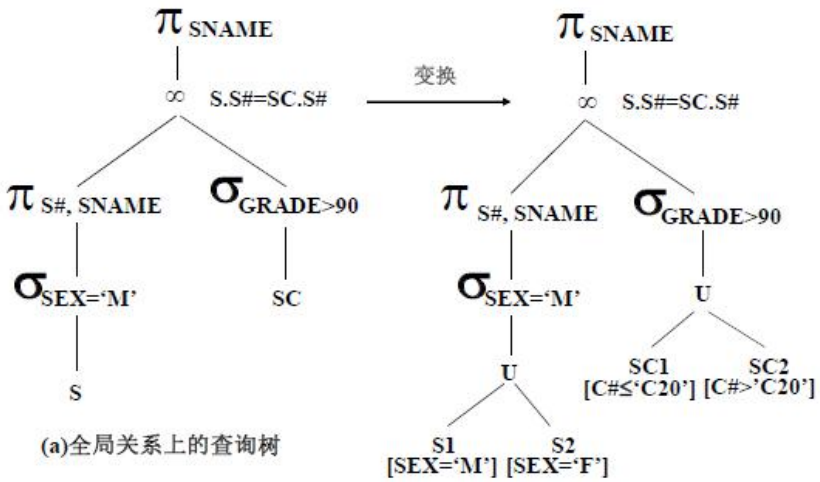
全局关系

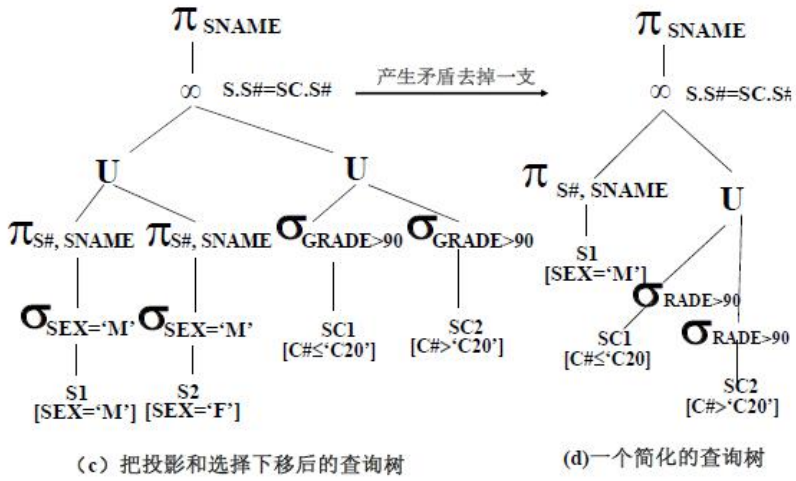
S(S#, SNAME, AGE, SEX)和SC(S#, C#, GRADE)被水平分片



查询问题：查找至少有一门功课成绩在90分以上的男生姓名

$\pi_{SNAME}(\sigma_{SEX='M' \text{ and } GRADE>90}(\sigma_{S.S\#=SC.S\#}(S \times SC)))$





（2）水平分片优化的基本思想

1. 尽量把选择条件下移到分片的限定关系处
2. 再把分片的限定关系与选择条件进行比较
3. 去掉它们之间存在矛盾的相应片断
4. 如果最后剩下一个水平片断，则重构全局关系的操作中，就可去掉“并”操作（至少可以减少“并”操作的次数）

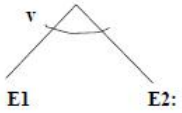
（3）垂直分片举例

全局关系

EMP(EMP#, ENAME, SALARY,DEPT#,DNAME)

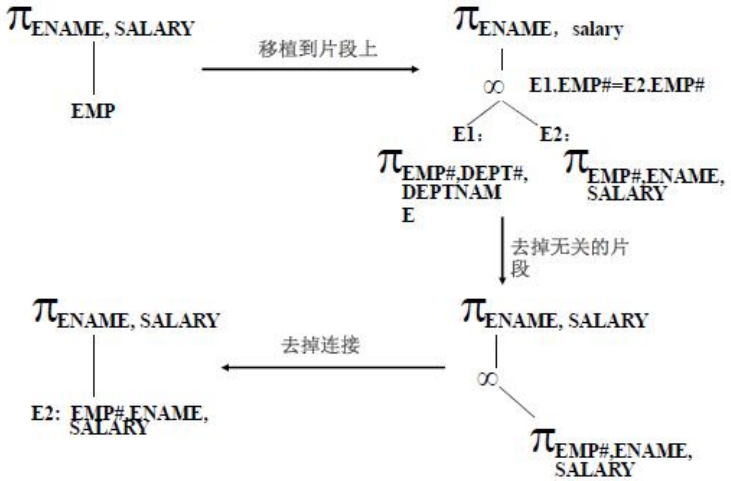
垂直分片：**E1 (EMP#,DEPT#,DNAME)**

EMP2(EMP#, ENAME, SALARY)



查询问题：雇员的姓名和工资情况

$\pi_{ENAME,SALARY}(EMP)$



（4）垂直分片的查询优化的基本思想

1. 把垂直分片所用到的属性集，与查询条件中的投影操作所涉及的属性相比较，去掉无关的垂直片断

2. 如果最后只剩下一个垂直片断与查询有关时，去掉重构全局关系的“连接”操作（至少可以减少“连接”操作的次数）

五、基于半连接算法的查询优化操作

1、半连接操作

- 假定有两个关系R,S,在属性R.A=S.B上做半连接操作，可表示为：
  - $R \propto_{A=B} S = \pi_R(R \infty_{A=B} S) = R \infty_{A=B} (\pi_B (S))$
  - $S \propto_{A=B} R = \pi_S(S \infty_{A=B} R) = S \infty_{A=B} (\pi_A (R))$
- 用半连接表示连接操作
  - $R \infty_{A=B} S = (R \propto_{A=B} S) \infty_{A=B} S$
  $= (R \infty_{A=B} (\pi_B (S))) \infty_{A=B} S$
  - $R \infty_{A=B} S = (S \propto_{A=B} R) \infty_{A=B} R$
  $= (S \infty_{A=B} (\pi_A (R))) \infty_{A=B} R$

（1）例一

R

A	B
2	a
10	b
25	c
30	d

S

A	C
3	x
10	y
15	z
25	w
32	x

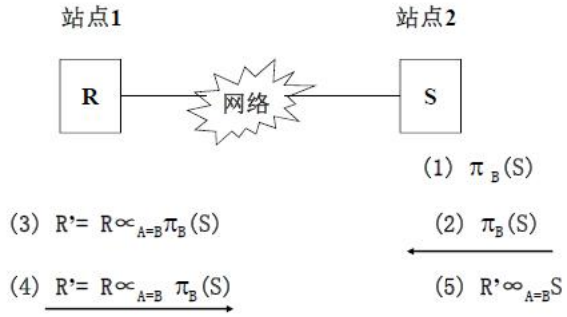
$R \propto_{A=A} S$

A	B
10	b
25	c

$S \propto_{A=A} R =$

A	C
10	y
25	w

2、半连接表示连接的代价估算



（1）关系的概貌

- Card(R) 片段关系 R 的元组数目
- Size(A) 属性 A 的大小(即字节数)
- Size(R) 片段关系的大小, 属性大小之和
- Val(A[R]) 属性 A 在 R 中出现的不同值的个数

（2）代数操作对关系概貌的影响

– 选择操作

$S = \sigma_F(R)$

Card(S)= ρ *Card(R)　　Size(S)=Size(R)

– 并操作

$T = R \cup S$

$Card(T) \leq Card(R)+Card(S)$

Size(T)=Size(R)=Size(S)

$Val(A[T]) \leq Val(A[R])+Val(A[S])$

– 连接操作

$T = R \infty S$

$Card(T) \leq (Card(R)*Card(S))/Val(A[R])$

Size(T) = Size(R)+Size(S)

$Val(A[T]) \leq Min(Val(A[R]), Val(A[S]))$　A 是连接属性

– 半连接

$T = R \propto S$

Card(T) = ρ *Card(R)

Size(T) = 第一个操作数Size（R）

Val(A[T]) = ρ *Val(A[R])

ρ ≈Val(A[S])/Val(Dom(A))

（3）代价公式：T=C0+C1*X

1. 在站点2上做投影π_B (S)
2. 把π_B (S)传到站点1上，代价为：
   - C_0+C_1* size (B)* val(B[S])
3. 在站点1上计算半连接，R'=R $\propto_{A=B}$ S
4. 把R'从站点1传到站点2的代价为：
   - C_0+C_1* size (R')* card(R')
5. 在站点2上执行连接操作：R' $\infty_{A=B}$S

（4）采用半连接的总代价

- $T_{半R}= 2C_0+C_1$* (size (R')* card(R') +size (B)* val(B[S]))
- $T_{半S}= 2C_0+C_1$* (size (S')* card(S') +size (A)* val(A[R]))

（5）比较 T 半 R 与 T 半 S，取最优者

3、半连接算法优化原理和步骤

（1）基本原理

1. 通常有两次传输
2. 但是传输的数据量和传输整个关系相比，要远远少
3. 一般有：T 半<<T 全
4. 半连接的得益：当 card（R）>>card（R'），可减少站点间的数据传输量
5. 半连接的损失：传输　B (S) =C0+C1* size (B)* val(B[S])
6. 基本原理是在传到另一个站点做连接前，消除与连接无关的数据，减少做连接操作的数据量，从而减小传输代价

（2）步骤

- 计算每种半连接方案的代价，并从中选择一种最佳方案
- 选择传输代价最小的站点，计算采用全连接的方案的代价
- 比较两种方案，确定最优方案

六、基于直接连接算法的查询优化

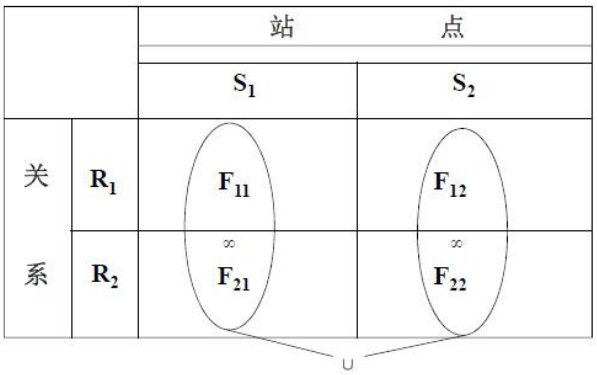
1、概述

（1）半连接算法和直接连接算法区别

- 取决于数据传输和局部处理的相对费用
- 如果传输费用是主要的，采用半连接
- 如果本地费用是主要的，采用直接连接

（2）优化算法（考虑关系分段）

- 利用站点依赖信息的算法



设关系R_i分片F_{i1}和F_{i2}, R_j分片F_{j1}和F_{j2}

关系R_i和R_j在属性A上满足条件

$F_{is} \infty_A F_{jt} = \varnothing$, 其中$s \neq t$,

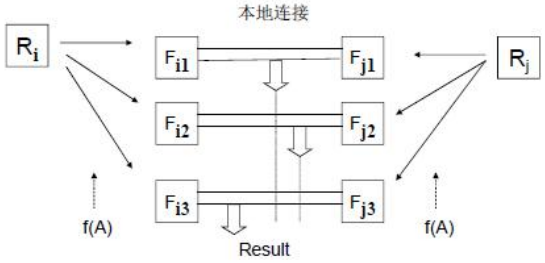
则称R_i和R_j在属性A上站点依赖

也就是说:

$R_i \infty_A R_j = U(F_{is} \infty_A F_{js})$, 对于包含着两个关系的片段的每个站点s都成立

此时关系的连接操作无站点间数据传输

$R_i \underset{A}{\infty} R_j$



* 推论：

– 1 若R_i和R_j在属性A上站点依赖，则R_i和R_j在任何包含A的属性集B上也站点依赖。

– 2 若R_i和R_j在属性A上站点依赖，另一属性（或属性组）B函数决定A，且$A \neq \varnothing$，则R_i和R_j在B上也站点依赖。

– 3 若R_i和R_j在属性A上站点依赖，且若R_j和R_k在属性B上站点依赖，则（$R_i \infty_A R_j \infty_B R_k$）=U（$F_{is} \infty_A F_{js} \infty_B F_{ks}$）；

查询$R_i \infty_A R_j \infty_B R_k$的连接操作能够以无数据传输的方式处理。

* 算法描述

– Placement_Dependency (Q, P, S)，其中：
  - R={R1,R2,R3,…,Rn}是查询Q引用的一组关系
  - P是站点依赖信息
  - S是一个连接操作可以无数据传输的执行的最大关系集合
  - 开始时S是空集。算法结束时，若S=R,则Q可以无数据传输执行
– 算法步骤
  - 初始化S= ∅， R={R1,R2,R3,…,Rn}
  - 若能找到一对关系R_i和R_j在属性A上站点依赖，且$R_i \infty_C R_j$包含在Q中，其中C包含A,那么把R_i和R_j放到S中，否则算法终止，返回空集S。
  - 只要存在R中而不在S中的关系R_k满足下面的特性，就把其放入S中：有S中的关系比如R_j，与R_k在属性B上有站点依赖关系，且$R_j \infty_B R_k$在Q中或者可以由Q导出，根据推论3，则R_k可被包含在S中。

– 分片与复制算法

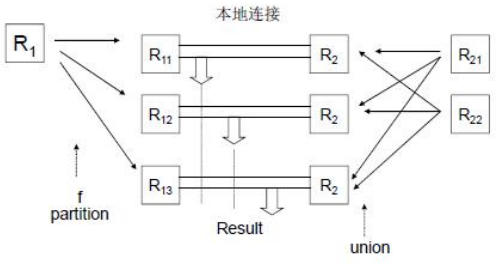
		站点	
		S_1	S_2
关系	R_1	F_{11}	F_{12}
	R_2	R_2	R_2

• 查询引用的某个关系的所有片段分布在这些站点上,其余被引用的关系复制到每一个选定的站点

R1 ∞ R2 = Ui (F1i ∞ R2)

• 算法可应用到涉及两个或两个以上的关系的查询
  – 其中一个关系保持分片状态
  – 其他关系可先连接起来，再被复制到各个站点
  – 在各个站点上，其他关系副本与相应的第一个关系的片断连接
  – 要求确定那个分片关系保留

$R_1 \infty R_2$



Fragmentation_and_replicate(Q, R, S)
For 每个保持分片状态的关系R_i
For 每个包含关系R_i的一个片段的站点S_j
计算在站点S_j执行子查询的完成时间
$FT(Q, S_j, R_i)$
计算关系Ri保持分片状态下的响应时间
$T_i = \max_j (FT(Q, S_j, R_i))$
选择$R_k = \min_i (T_i)$为保持分片状态的关系

** 举例：

已知 R1 分段 F11 和 F12 的大小为: |F11|=|F12|=50

R2 分段 F21 和 F22 的大小为: |F21|= 100 |F22|=200

设数据通讯 C0=0, C1=1 （参照半连接优化定义）

本地连接 Cost=J(x1, x2)=5*(x1+x2)

并操作 Cost = U(x1, x2) = 2*(x1+x2)

令 R1 保持分片状态，则：站点 S1 的完成时间

FT(Q, S1, R1) = 200+2*(100+200)+5*(50+300)=2550

同理: FT(Q, S2, R1) = 100+2*(100+200)+5*(50+300)=2450

因此，查询响应时间在 R1 保持分片状态为 2550.

令 R2 保持分片状态，则：站点 S1 的完成时间

FT(Q, S1, R2) = 50+2*(50+50)+5*(100+100)=1250

同理:

FT(Q, S2, R2) = 50+2*(50+50)+5*(200+100)=1750

因此，查询响应时间在 R2 保持分片状态为 1750.

因为:

R1 保持分片状态的响应时间>R2 保持分片状态的响应时间

所以: 选择 R2 保持分片计算查询

- Hash 划分算法

* 定义：利用 hash 函数对分片关系上的连接属性作站点依赖计算，再根据此分片，获取站点依赖的连接算法。

例如, 运用Hash函数

$$h(a) = \begin{cases} 1 & \text{若a 是奇数} \\ 0 & \text{若a 是偶数} \end{cases}$$

对R中每个元组, h(a)为1送入站点S_1, h(a)为0送入站点S_2. 于是片段关系R被划分为R^o和R^e

$$R_1 \infty R_2 = (R_1^o \infty R_2^o) \cup (R_1^e \infty R_2^e)$$

- 利用Hash函数对分片关系上的连接属性作站点依赖计算，再据此分片，以获取站点依赖的JN算法
- 例如，运用Hash函数

$$h(a)=\begin{cases}1 & \text{若a是奇数}\\ 0 & \text{若a是偶数}\end{cases}$$

- 片断F_{11}按属性A的值为奇和偶数划分成F_{11}^{o}和F_{11}^{e}，片断F_{12}划分成F_{12}^{o}和F_{12}^{e}
- 站点S_2上$F_{12}'=F_{11}^{e}\cup F_{12}^{e}$，站点$S_1$上$F_{11}'=F_{11}^{o}\cup F_{12}^{o}$
- 显然$\pi_A(F_{11}')$和$\pi_A(F_{12}')$没有公共值，前面是奇数值后面是偶数值
- $F_{12}'\propto F_{11}'$是空集，这说明R_1和R_2在新组成的片断下在属性A上站点依赖。
- $R_1 \propto R_2 = (F_{11}'\propto F_{21}') \cup (F_{12}'\propto F_{22}')$

考察三个关系R_1，R_2和R_3，它们在两个站点上，有两种情况：

– 在同一属性A上连接，$R_1\propto_A R_2\propto_A R_3$
  - 在三个关系的片断上应用Hash函数
  - 使用新组建的片断，三个关系在属性A上将满足站点依赖
  - 经这种划分和数据传送之后，两个站点上的片断在属性A上的连接就可以并行进行，合并执行结果给出答案

– 在不同属性上连接，$R_1\propto_A R_2\propto_B R_3$

① 在同一属性 A 上连接：$R_1\propto_A R_2\propto_A R_3$

② 在不同属性上连接：$R_1\propto_A R_2\propto_B R_3$

问题：

• 在属性 A 上应用同样的 Hash 函数，在属性 B 上也应用同样的 Hash 函数，可能得不到希望的站点依赖

• 因 R1 中属性 A 的值是奇数的发往 S1，R3 中属性 B 是奇数的元组发往 S1.但 R2 中某些元组可能在 A 上有奇数值，而在 B 上有偶数值

• 解决方法是允许这些元组在两个站点上都存在。

$R1\propto_A R2\ \propto_B R3$

若R1与R2在A上有相同的Hash函数，R2与R3在属性B上有相同的Hash函数

S1	S2
$F^{0}_{11}(A)$	$F^{e}_{12}(A)$
$F^{0}_{31}(B)$	$F^{e}_{32}(B)$

2、算法比较（两个关系）

- 站点依赖算法
  - 无数据传递
  - 可利用索引做本地连接
  - 每个站点连接数据总量是 R，两个片段
- 分片和复制算法
  - 数据传输总量是 R
  - 每个站点的连接数据量是(3/2)R,一个全关系和一个片断
- Hash 划分算法
  - 数据传送量是 R
  - 每个站点的连接数据量同站点依赖

** 练习 1：

有关系**R**，**S**，**T**，如图所示

1. 计算连接**R** ∞ **S** ∞ **T**
2. 计算半连接**R**∝**S**，**S**∝**R**，**S**∝**T**，**T**∝**S**，**R**∝**T**，**T**∝**R**

R

A	B	C
2	3	5
5	3	6
1	6	8
3	4	6
5	3	5
2	6	8

S

B	C	D
3	5	6
3	5	9
6	8	3
5	9	6
4	1	6
5	8	4

T

D	E	F
6	6	9
8	7	8
8	5	6
3	8	9

** 练习 2：

在如下R, S的概貌上计算R $\infty_{A=B}$ S

Size(R)=50, Card(R)=100, Val(A[R])=50, Size(A)=3

Size(S)=5, Card(S)=50, Val(B[S])=50, Size(B)=3

R $\propto_{A=B}$ S 的选择度 $\rho = 0.2$

S $\propto_{A=B}$ R 的选择度 $\rho = 0.8$

C_0=0,C_1=1

问:

1. 使用 $\propto$ 简化程序在R的站点执行∞
2. 使用 $\propto$ 简化程序在S的站点执行∞
3. 使用直接连接在R站点执行∞
4. 使用直接连接在S站点执行∞

那种方案较优?

解:

1. $COST1 = 2C0 + C1(Size(A)\cdot Val(A[R]) + Size(S)\cdot Card(S)\cdot \rho_{S\,SJ\,R})$
   $= 2C0 + C1(3\cdot 50 + 5\cdot 50\cdot 0.8) = 2C0 + 350C1$
2. $COST2 = 2C0 + C1(Size(B)\cdot Val(B[S]) + Size(R)\cdot Card(R)\cdot \rho_{R\,SJ\,S})$
   $= 2C0 + C1(3\cdot 50 + 50\cdot 100\cdot 0.2) = 2C0 + 1150C1$
3. $COST3 = C0 + C1\cdot Size(S)\cdot Card(S)$
   $= C0 + C1\cdot 5\cdot 50 = C0 + 250C1$
4. $COST4 = C0 + C1\cdot Size(R)\cdot Card(R)$
   $= C0 + C1\cdot 50\cdot 100 = C0 + 5000C1$

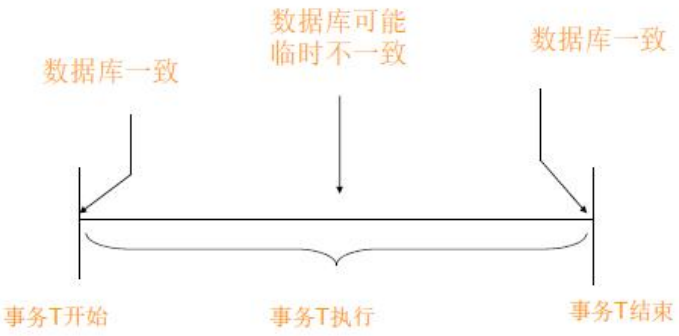
所以第三种方案最优。

第六讲 分布式数据库中的事务管理和恢复

一、分布式事务概述

1、定义和特性

（1）事务概念：事务是访问或更新各种数据项的最小逻辑工作单位，它是一个操作序列。事务提交时数据库必须是一致的。



* 集中式：事务和操作数据在一个站点上；不存在传输费用
* 分布式：操作数据分布在不同的站点上；事务也在多个站点上执行；站点和通信链路故障都可能导致错误发生；分布式事务的恢复复杂的多

（2）事务分类

* 全局事务
• 通常由一个主事务和在不同站点上执行的子事务组成
• 主事务：负责事务的开始、提交和异常终止
• 子事务：完成对相应站点上的数据库的访问操作
* 局部事务
• 仅访问或更新一个站点上的数据的事务

（3）ACID 特性

– 原子性（Atomicity）

事务的操作要么全部执行，要么全部不执行,保证数据库一致性状态

– 一致性（Consistency）

事务的正确性，串行性。并发执行的多个事务,其操作的结果应与以某种顺序串行执行这几个事务所得的结果相同

– 持久性（Durability）

当事务提交后，其操作的结果将永久化，而与提交后发生的故障无关

– 隔离性（ Isolation）

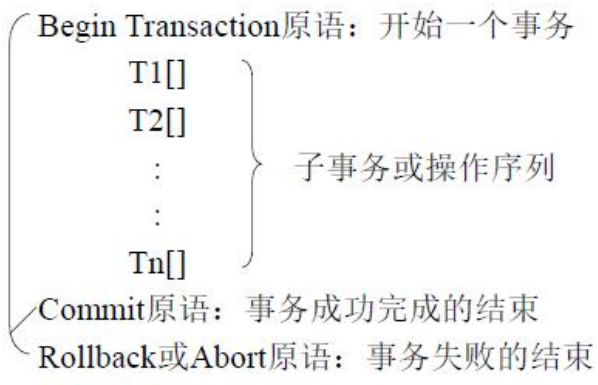
虽然可以有多个事务同时执行，但是单个事务的执行不应该感知其他事务的存在，因此事务执行的中间结果应该对其他并发事务隐藏

** 注意：全局事务的主事务和子事务全部成功提交，才能改变数据库状态，有一个失败，其他子事务操作都要撤销。

** 举例：从账号 A 向账号 B 转账$50:

1. read(A)
2. A := A – 50
3. write(A)
4. read(B)
5. B := B + 50
6. write(B)

（4）分布式事务的一般结构



（5）分布式事务的状态

• 活动从事务开始执行的初始状态始， 事务执行中保持该状态

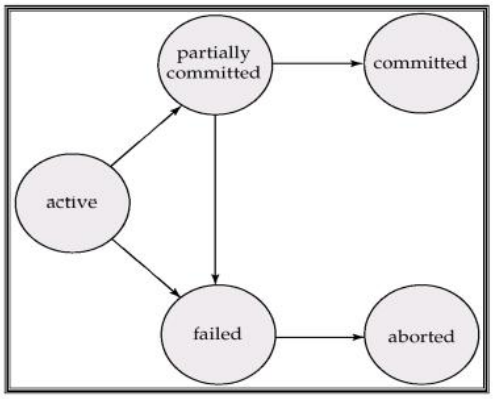
• 部分提交事务的最后一个语句执行后进入该状态.

• 失败一旦发现事务不能正常执行时进入该状态

• 夭折当事务被回滚后，数据库恢复到事务开始执行前的状态。事务夭折后有两种选择

- 重启动仅当没有内部逻辑错误时
- 杀死

• 提交当事务成功执行后.



** 实例：转账应用

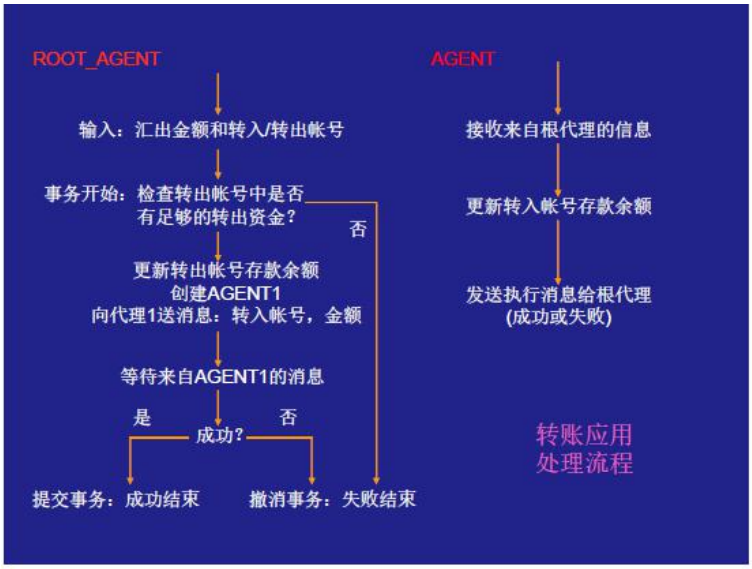
事务在两个账户之间执行“基金汇兑”操作。如果汇兑的金额小于转出帐号现有金额，就撤销如果大于等于就提交。

全局关系：Account (Account-number, Amount)

假设账户分布在网络的不同站点上。

FUND_TRANSFER:

```
read (terminal,$AMOUNT,$FROM_ACC,$TO_ACC);
begin_transaction;
select AMOUNT into $FROM_AMOUNT from ACCOUNT
    where ACCOUNT_NUMBER=$FROM_ACC;
if $FROM_AMOUNT-$AMOUNT<0 then abort
else begin
        update ACCOUNT
            set AMOUNT = AMOUNT-$AMOUNT
            where ACCOUNT_NUMBER = $FROM_ACC;
        update ACCOUNT
            set AMOUNT = AMOUNT+$AMOUNT
            where ACCOUNT_NUMBER = $TO_ACC;
        commit
end
```



转账事务的两个代理：

```
ROOT_AGENT:
        read(terminal,$AMOUNT,$FROM_ACC,$TO_ACC);
        begin_transaction
                select AMOUNT into $FROM_AMOUNT from ACCOUNT
                        where ACCOUNT_NUMBER=$FROM_ACC;
                if $FROM_AMOUNT-$AMOUNT<0 then abort
                else begin
                    update ACCOUNT
                    set AMOUNT = AMOUNT-$AMOUNT
                     where ACCOUNT_NUMBER = $FROM_ACC;
        create AGENT;
        send to AGENT($AMOUNT,$TO_ACC);
         wait()
        commit
   end
AGENT:
   receive from ROOT_AGENT($AMOUNT,$TO_ACC);
   update ACCOUNT set AMOUNT=AMOUNT+$AMOUNT where
        ACCOUNT=$TO_ACC;
   send to ROOT_AGENT('SUCCESS'/'FALL')
```

2、分布式事务管理的问题和目标

（1）问题

- 处理数据项的多个副本
  - 分布式事务处理负责保持同一数据的多个副本之间的一致性。
- 单个站点的故障
  - 一个站点或多个站点故障时，DDBMS 继续与其他正常运行的站点一起继续工作
  - 当故障站点恢复时，DDBMS 协同故障站点的 DBMS，使该站点与系统连接时，局部数据库与其他站点同步
- 通信网络的故障
  - 必须能够处理两个或者多个站点间的通信网络故障
- 分布式提交
  - 如果提交分布式事务过程中有一个站点发生故障，提交就会产生问题

- 两阶段提交协议用于解决这一问题

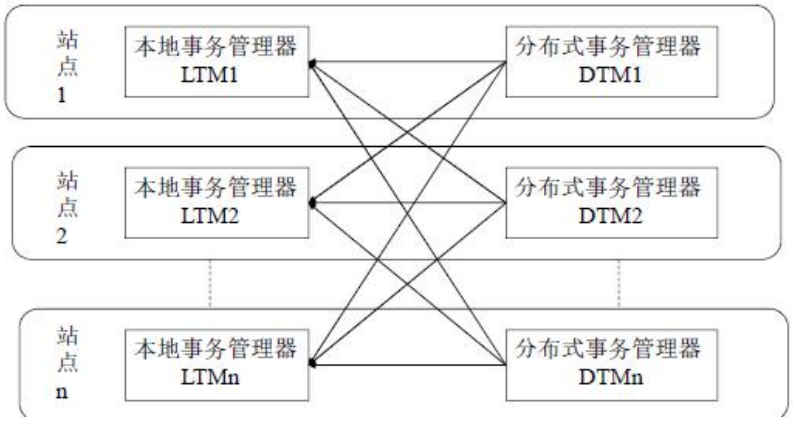
（2）目标

- 事务能有效、可靠、并发的执行
- 维护事务的 ACID 性质
- 获得最小的主存和 CPU 开销，降低报文数目，加快响应时间
- 获得最大限度的可靠性和可用性

二、分布式事务的执行与恢复

1、分布式事务管理的抽象模型

（1）抽象模型



（2）事务管理

* DTM 功能

- 保证分布式事务 ACID 特性，

• 特别是原子性，使每一站点的子事务都成功执行，或者都不执行。

• 通过向各站点发 begin-transaction,commit 或者 abort,create 原语来实现的

- 负责协调由该站点发出的所有分布式事务的执行

• 启动分布式事务的执行

• 将分布式事务分解为子事务，并将其分派到恰当的站点上执行

• 决定分布式事务的终止（子事务都提交或者都撤销）

- 支持分布式事务执行位置透明性

• 实现了对网络上各站点的各子事务的监督和管理

• 完成对整个分布式事务执行过程的调度和管理

• 保证分布式数据库系统的高效率

* LTM 功能

- 保证本地事务的 ACID 特性
- 维护一个用于恢复的日志，代替 DTM 把分布事务的执行与恢复信息记入日志
- 参与适当的并发控制模式，以协调在该站点上执行的事务的并发执行。

2、分布式数据库系统中的故障

（1）站点故障

- 事务故障

• 由非预期的、不正常的程序结束所造成的故障，如：计算溢出、完整性破坏、操作员干预、输入输出错误、死循环等）

• 处理方法：内存、磁盘上信息没有损失，使用 Log 做 Rollback

- 系统故障

• 造成系统停止运行的任何事件，要求系统重启动，如 CPU 出错、缓冲区满、系统崩溃等

• 处理方法：内存、I/O Buffer 内容皆丢失，DB 没有破坏，恢复时，搜索 Log，确定 Rollback 的事务。

- 介质故障：

• 辅助存储器介质遭破坏

• 处理方法：如数据丢失，日志无损失，从某个 Dump 状态开始执行已提交事务；数据与日志都丢失不可能完全恢复。

（2）通讯故障

• 报文故障

- 报文错，报文失序、丢失、延迟

• 网络分割故障（网络断连）

通讯发生，某个报文 Message 从 Site x 发往 Site y，正常情况:

(a) 在某时间段 Dmax 之前, x 站点收到 y 发回的应答信息(Ack)

(b) y 收到的 Message 是一个合适的次序

(c) Message 本身的信息是正确的

但是当某个 Dmax 之后, x 还没收到 y 的 Ack，则可能发生:

(a) Message 或 Ack 信息丢失

(b) 网络分割，即网络不通

3、事务故障恢复的基本概念

• 事务恢复

- 当发生故障时，保证事务原子性的措施称为事务故障恢复，简称事务恢复

- 主要依靠日志来实现

• 事务状态转移跟踪（操作）

- Begin_transaction：标记事务开始执行

- Read & write：表示事务对某个数据项进行读写
- End_transaction：表示读写操作已完成，标记事务执行结束
- Commit_transaction：表示事务已经成功结束，任何改变已不可更改
- Rollback (abort)：表示事务没有成功结束，撤销事务对数据库所作的任何改变

• 事务的提交点
- 当事务 T 所有的站点数据库存取操作都已成功执行；
- 所有操作对数据库的影响都已记录在日志中。到达提交点
- 提交点后事务就成为已提交的事务，并假定其结果以永久记录在数据库中
- 事务在日志中写入提交记录[commit,T]
- 在系统发生故障时，需要扫描日志，检查日志中写入[start_transaction,T],但没有写入[commit,T]的所有事务 T
- 恢复时必须回滚这些事务以取消他们对数据库的影响
- 此外，还必须对日志中记录的已提交子事务的所有写操作进行恢复。

• 事务的提交点相关操作
- 日志文件保存到磁盘上
- 一般先将文件的相关块，从磁盘拷贝到主存的缓冲区，然后更新，再写回磁盘
- 缓冲区中会经常存在一个或多个日志文件块，写满后一次性写回磁盘
- 系统崩溃时，主存中的信息会丢失，这些信息无法利用
- 因此，事务到达提交点之前，未写到磁盘的日志必须写入，称为事务提交前强制写日志。（强制写入机制的效率如何？）

• 日志
- Log：记录所有对 DB 的操作
- 事务标识：每个事务给定一个具有惟一性的标识符
- Log 记录项:

• [start_transaction, T],
• [write_item, T, x, 旧值, 新值]
• [read_item, T, x]
• [commit, T]
• [abort, T]

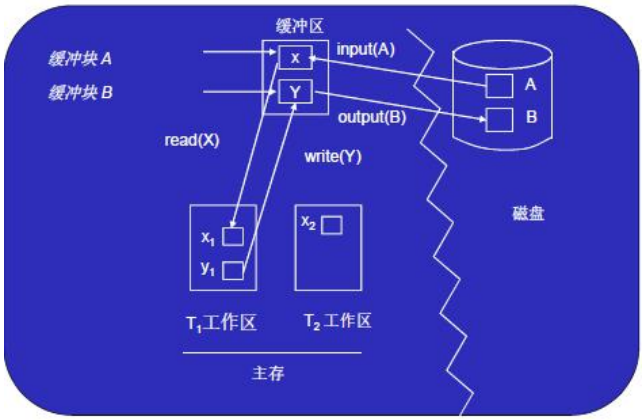
- 写动作：写 Log 比写数据优先
- Log 存储：一般存在盘上，还会定期备份到磁带上

Log举例

Log	Write	Output
<start, T_0>		
<write, T_0, A, 1000, 950>		
<write, T_0, B, 2000, 2050>		
	$A = 950$	
	$B = 2050$	
<commit, T_0 >		
<start, T_1 >		
<write, T_1, C, 700, 600>		
	$C = 600$	
		B_B, B_A
<commit, T_1 >		
		B_C

注: B_X 表示含有X的存储块.

- 数据访问



- 检查点（Checkpoint）
  - 设置一个周期性（时间/容量）操作点

a) Log Buffer 内容写入 Log 数据集

b) 写检查点 Log 信息：当前活动事务表，每个事务最近一次 Log 记录在 Log 文件中的位置

c) DB Buffer 内容写入 DB

d) 将本次检查点 Log 项在 Log 文件中的地址记入“重启动文件”

4、事务故障的恢复

- 事务本身也会发生故障，也是主要通过日志来实现恢复
- 恢复原则
  - 孤立和逐步退出事务的原则

undo 事务已对 DB 的修改(不影响其他事务的可排除性局部故障，如事务操作的删除、超时、违反完整性原则、资源限制和死锁等)

  - 成功结束事务原则

Redo 已成功事务的操作

  - 夭折事务原则

撤销全部事务，恢复到初态，两种做法：利用数据备份和 Undo

- 本地事务恢复(与集中式恢复相同)

– 从“重启动文件” 读出最近 Checkpoint 的地址，并定出 Checkpoint 在 Log 文件中的位置

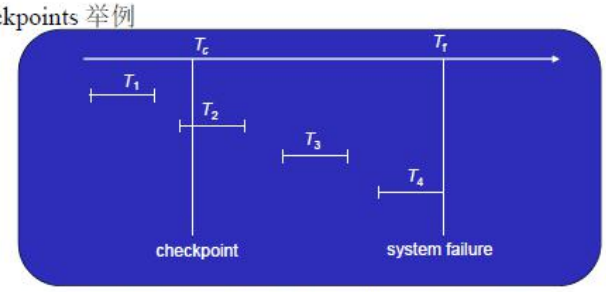
– 创建 Redo 表（初态为空）, Undo 表(即 Checkpoint 相应内容中的活动事务表)

– 检查得出 Undo 事务（向前扫描，遇到 begin transaction 的 log 记录，其对应的事务）与 Redo 事务（向前扫描，遇到 commit 的 log 记录，其对应事务）

– 反向扫描 Log，将 Undo 表中事务回滚，直到遇到对应的 Begin Trans

– 正向扫描 Redo 事务的 Log 记录，并执行之，直到对应的 Commit 记录

Checkpoints 举例



- T_1 可以忽略 (因为有检查点，更新已经被写入磁盘)
- T_2 和 T_3 redo.
- T_4 undo

三、两阶段提交协议

1、基本思想和内容

- 基本思想

– 将本地原子性提交行为的效果扩展到分布式事务，保证了分布式事务提交的原子性，并在不损坏 Log 的情况下,实现快速故障恢复，提高 DDB 系统的可靠性.

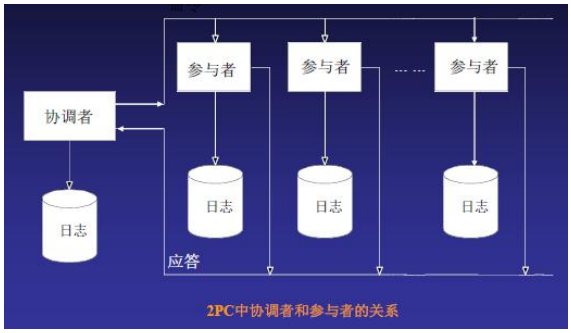
– 第一阶段：表决阶段

– 第二阶段：执行阶段

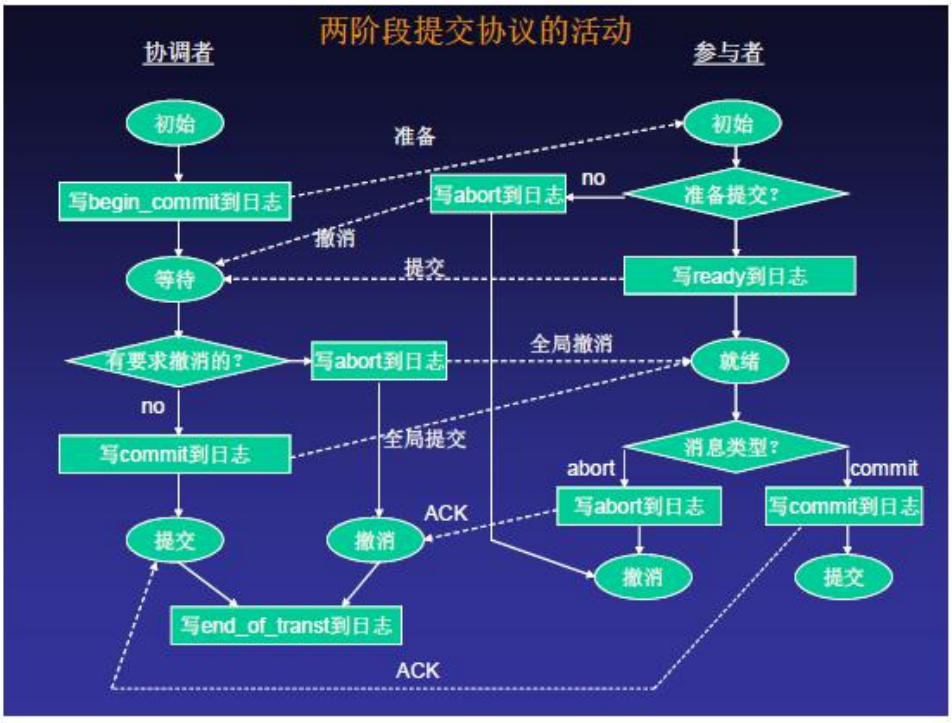
- 两类代理

– 协调者(Coordinator)：提交和撤销事务的决定权，一般是总代理

– 参与者(Participants)：负责在本地数据库中执行写操作，并且向协调者提出提交和撤销子事务的意向



• 表决阶段
– 目的是形成一个共同的决定
– 首先，协调者给所有参与者发送“准备”消息，进入等待状态
– 其次，参与者收到“准备”消息后，检查是否能够提交本地事务
• 如能，给协调者发送“建议提交”消息,进入就绪状态
• 如不能，给协调者发送“建议撤销”消息，可以单方面撤销
– 第三，协调者收到所有参与者的消息后，他就做出是否提交事务的决定
• 只要有一个参与者投了反对票，就决定撤销整个事务，发送“全局撤销”消息给所有参与者，进入撤销状态
• 否则，就决定提交整个事务，发送“全局提交”消息给所有参与者，进入提交状态
• 执行阶段
– 实现表决阶段的决定，提交或者撤销

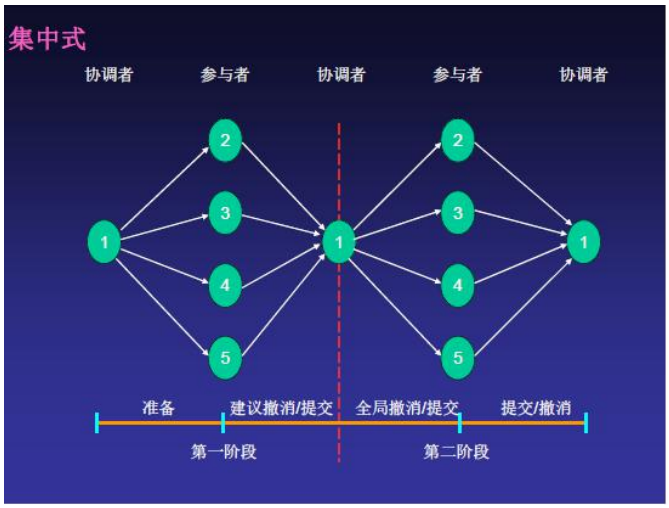


• 2PC 协议的重要特点
– 允许参与者单方面撤销事务
– 一旦参与者确定了提交或撤销协议，它就不能再更改它的提议
– 当参与者处于就绪状态时，根据协调者发出的消息种类，它可以转换为提交状态或者撤销状态
– 协调者根据全局提交规则做出全局终止决定
– 协调者和参与者可能进入互相等待对方消息的状态，使用定时器，保证退出

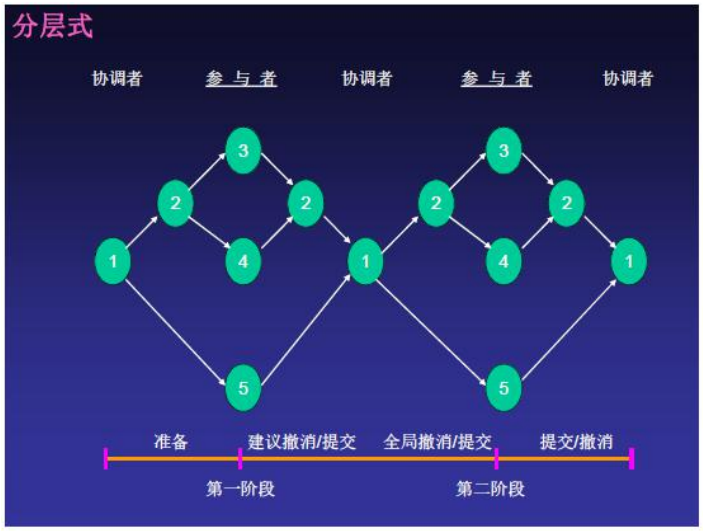
消息等待状态

2、通信结构

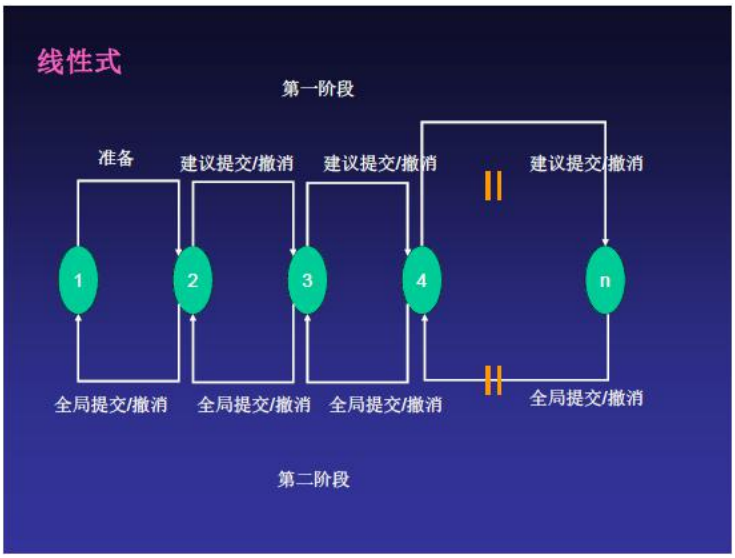
- 集中式
  - 通讯只发生在协调者和参与者之间，参与者之间不交换信息



- 分层式
  - 协调者是在树根的 DTM 代理者，协调者与参与者之间的通讯不用直接广播的方法进行，而是使报文在树中上下传播。每个 DTM 代理是通信树的一个内部节点，它从下层节点处收集报文或向它们广播报文。

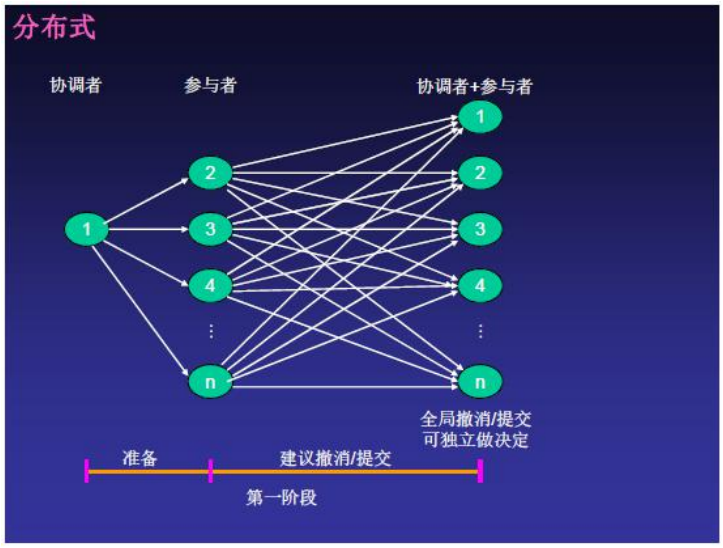


- 线性
  - 参与者之间可以互相通信。系统中的站点间要排序，消息串行传递。支持没有广播功能的网络



• 分布式

– 允许所有参与者在第一阶段相互通信，从而可以独立做出事务终止决定。



3、两阶段提交协议和故障恢复

（1）站点故障

a> 参与者在将“Ready”记录入 Log 之前故障

• 此时协调者(C)达到超时，Abort 发生。站点（P）恢复时，重启动程序将执行 Abort，不必从其他站点获取信息。

b> 当将“Ready”写入 Log 后，站点故障

• 此时所有运行的站点都将正常结束事务（Commit/Abort)。P 恢复时，因为 P 已 Ready，所以不可判定 C 的最终决定。因此恢复时，重启动程序要询问 C 或其他站点。

c> 当 C 将“Prepare”写入 Log，但“G-commit”/”G-abort”还没有写入前故障

• 所有回答“Ready”的 P 等待 C 恢复。C 重启动时，将重开提交协议，重发“Prepare”，于是 P 要识别重发。

d> C 在将“G-commit”/”G-abort”写入 Log 后，“end_of_transt”没有写入前故障

• 收到命令的 P 正常执行，C 重启动程序必须再次向所有 P 重发命令。以前没有收到命令的 P 也必须等待 C 恢复，P 要识别两次命令。

e> “end_of_transt”写入 Log 后故障

• 无任何动作发生

（2）报文丢失

a> 从 P 发出的“Ready”/“Abort”报文丢失

• C 达到超时，整个事务执行“G-abort”。

b> “Prepare”报文丢失

• P 等待，C 得不到回答，结果同 2.a>

c> “G-commit”/”G-abort”报文丢失

• P 处于不确定状态。回答“Abort”的可以确定其工作，回答“Ready”的不行。此时可以修改加入计时器，超时则申请重发命令。

d> “Ack”报文丢失

• C 超时，可重发“G-commit”/”G-abort”命令，P 无论是否有活动，都重发“Ack”报文

（3）网络分割

站点假设分成两组：协调者组和参与者组。

• 一组是协调者，一组是参与者。于是从协调者看是参与者组故障。从参与者组看是协调者站点故障。其动作按产生网络分割时的状态，类似站点故障处理。

四、分布式数据库中的数据更新

1、多站点数据更新

– 方法：站点 A 上有事务 T 对 X 更新, X 在 B1,…Bn 和 C1,…Cm 上有副本, 则也要对这些副本更新

– 问题

• 多个站点同时更新不现实(每一个站点某一时刻与站点 A 连通的概率小于 1）

• 对未连通站点上的副本更新时出错, 更新的顺序也不一定是连通顺序

2、主文本更新

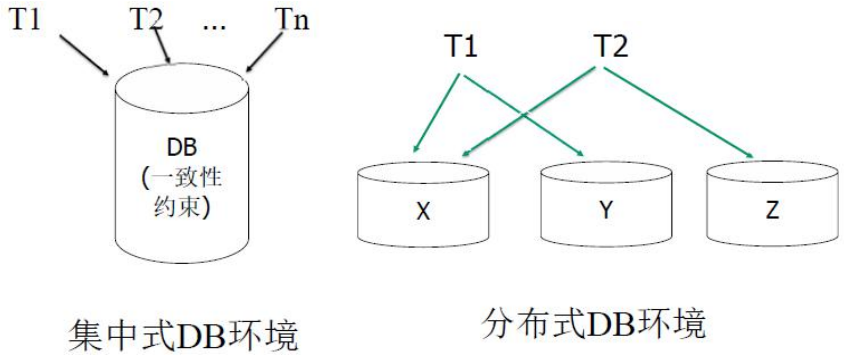
– 思想：指定主副本，修改只对主副本进行。更新传播到辅助副本时，也按在主副本上执行的更新顺序执行
– 问题
• 修改传播必须在短时间内完成，否则将获得“过时”数据
• 主副本不可用，引得其他副本也不可用
– 常用方法
• 移动主文本法
– 如果主站点此时尚未连通，则另选一个辅站点中的辅文本为该数据新的主文本进行更新
– 待原主文本站点连通后，系统将自动把它改为辅文本，并按记录要求执行更新
– 如果更新在主文本上进行，但主文本站点与网络未接通，则此次更新操作失败，事务被撤销，因为无法传播更新
• 移动文本法的问题
• 网络分割成很多部分时，更新处理会不一致
• 网络分割成 W1,W2，W1 中 X 更新为 R, W2 中 X 更新为 S,网络连通时，使用 R 还是 S 来恢复 X 呢？

3、快照及其更新
– 与视图相似，是导出的关系
– 快照的数据是实际存放在数据库中的，视图不是
– 用于某些需要“冻结”数据的应用
** 举例：
Define Snapshot HP-Book as SELECT * FROM Book WHERE Price>$100 REFRESH Every day
– 快照不考虑数据的辅助副本，只关心主副本和这个主副本上定义的多个快照
– 快照与视图一样可以定义为一个或多个主副本的部分或全部
– 查询操作可使用快照，也可使用主副本，对更新操作还是在主副本上进行
– 主文本更新时刷新快照

第七讲 分布式数据库中的并发控制
一、并发控制的概念和理论
1、概念
* 事务的并发操作：数据库总有若干个事务在运行，这些事务可能并发的存取相同的数据。当数据库中有多个事务并发执行时，系统必须对并发事务之间的相互

作用加以控制，这是通过并发控制机制来实现的。

* 分布式并发控制的重要性：分布式数据库中的并发控制解决多个分布式事务对数据并发执行的正确性，保证数据库的完整性和一致性。比集中式并发控制更复杂。



集中式DB环境　　分布式DB环境

* 并发控制的问题

（1）丢失更新

时间	更新事务T1	数据库中X的值	更新事务T2
t_0		100	
t_1	FIND x		
t_2			FIND x
t_3	x:=x-30		
t_4			x:=x*2
t_5	UPDATE x		
t_6		70	UPDATE x
t_7		200	

注：其中FIND表示从数据库中读值，UPDATE表示把值写回到数据库
T1T2，结果140，T2T1,结果170，
得到结果是200，显然是不对的，T1在t_7丢失更新操作。

（2）不一致分析

时间	更新事务T1	数据库中A的值	更新事务T2
t_0		100	
t_1	FIND x		
t_2			FIND x
t_3	x:=x-30		
t_4	UPDATE x		
t_5		70	

注：在时间t5事务T2仍认为x的值是100

（3）读脏数据

时间	更新事务T1	数据库中A的值	更新事务T2
t_0		100	
t_1	FIND x		
t_2	x:=x-10		
t_3	UPDATE x		
t_4		90	FIND x
t_5	ROLLBACK		
t_6		100	

注： 事务T2依赖于事务T1的未完成更新

2、事务可串行化理论

- 调度
  - 一组事务的调度必须包含这些事务的所有操作
  - 调度中某个事务的操作顺序必须保持与该事务原有的顺序相同
- 串行调度
  - 一个事务的第一个动作是在另一个事务的最后一个动作完成后开始. 即调度中事务的各个操作不会交叉, 每个事务相继执行.
- 一致性调度
  - 调度可以使得数据库从一个一致性状态转变为另一个一致性状态，则称调度为一致性调度
- 可串行化调度
  - 如果一个调度等价于某个串行调度，则该调度称为可串行化调度。
  - 也就是说，该调度可以通过一系列非冲突动作的交换操作使其成为串行调度

3、并发控制机制的常用方法及其分类

- 保证只产生可串行化调度的机制
- 并发控制机制分类
  - 建立在相互排斥地访问共享数据基础上的算法
  - 通过一些准则(协议)对事务进行排序的算法
  - 悲观并发控制法（事务是相互冲突的），
    乐观并发控制法（没有太多的事务相互冲突）

二、分布式数据库系统并发控制机制的封锁技术

1、基于封锁的并发控制方法概述

（1）基本思想

事务访问数据项之前要对该数据项封锁，如果已经被其他事务锁定，就要等待，

直到哪个事务释放该锁为止。

（2）锁的粒度

- 锁定数据项的范围
- 数据项层次

• 一条数据库记录
• 数据库记录中的一个字段值
• 一个磁盘块（页面）
• 一个完整的文件
• 整个数据库

（3）粒度对并发控制的影响

- 大多数 DBMS 缺省设置为记录锁或页面锁
- 粒度小，并发度高，锁开销大

• 数据项比较多，锁也多，解锁和封锁操作多，锁表存储空间大

- 粒度大，并发度低，锁开销小

• 如果是磁盘块，封锁磁盘块中的一条记录 B 的事务 T 必须封锁整个磁盘块
• 而另外一个事务 S 如果要封锁记录 C，而 C 也在磁盘块中，由于磁盘块正在封锁中，S 只能等待
• 如果是封锁粒度是一条记录的话，就不用等待了

（4）如何确定粒度

- 取决于参与事务的类型
- 如果参与事务都访问少量的记录，那么选择一个记录作为粒度较好
- 如果参与事务都访问同一文件中大量的记录，则最好采用块或者文件作为粒度

（5）锁的类型

- 共享锁：Share 锁，S 锁或者读锁
- 排它锁：Exclusive 锁，X 锁，拒绝锁或写锁。
- 更新锁：Update 锁，U 锁

** 补充：更新 (U) 锁可以防止通常形式的死锁。一般更新模式由一个事务组成，此事务读取记录，获取资源（页或行）的共享 (S) 锁，然后修改行，此操作要求锁转换为排它 (X) 锁。如果两个事务获得了资源上的共享模式锁，然后试图同时更新数据，则一个事务尝试将锁转换为排它 (X) 锁。共享模式到排它锁的转换必须等待一段时间，因为一个事务的排它锁与其它事务的共享模式锁不兼容；发生锁等待。第二个事务试图获取排它 (X) 锁以进行更新。由于两个事务都要转换为排它 (X) 锁，并且每个事务都等待另一个事务释放共享模式锁，因此发生死锁。若要避免这种潜在的死锁问题，请使用更新 (U) 锁。一次只有一个事

务可以获得资源的更新 (U) 锁。如果事务修改资源，则更新 (U) 锁转换为排它 (X) 锁。否则，锁转换为共享锁。

（6）锁的选择
- 数据项既可以读也可以写.则要用 X 锁
- 如果数据项只可以读.则要用 S 锁.

（7）锁的操作
- Read_lock(x):读锁
- Write_lock(x)：写锁
- unlock(x)：解锁

（8）数据项的状态
- read_locked: 读锁
- Write_locked:写锁

（9）具体操作方法
- 锁表中每条记录有四个字段：<数据项名称，锁状态，读锁的数目，正在封锁该数据项的事务>

（10）锁的转换

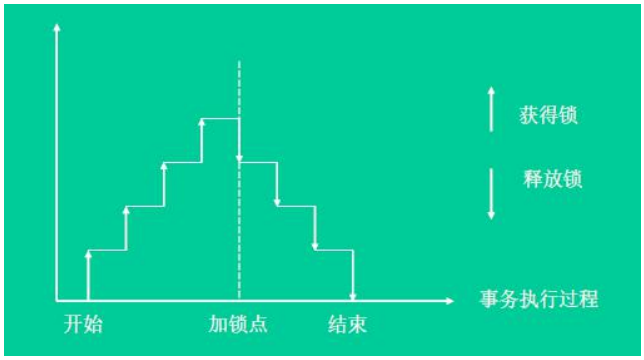
1. 特定条件下，一个已经在数据项 x 上持有锁的事务 T ，允许将某种封锁状态转换为另外一种封锁状态
2. 比如，一个事务先执行了 read_lock（x）操作，然后他可以通过执行 write_lock(x) 操作来升级该锁
3. 同样，一个事务先执行了 write_lock(x) 操作，然后他可以通过执行 read_lock（x） 操作来降级该锁

** 注意：即使满足封锁规则也不能保证能产生串行化调度，串行调度的结果也不一定相同。

分布式数据库基本封锁算法
- 简单的分布式封锁方法
  - 类似集中式，将同一数据的全部副本封锁，然后更新，之后解除全部封锁
  - 缺点是各站点间进行相当大的数据传输，如果有 N 个站点，就有：
- N 个请求封锁的消息 N 个封锁授权的消息
- N 个更新数据的消息 N 个更新执行了的消息
- N 个解除封锁的消息
- 主站点封锁法
  - 定义一个站点为主站点，负责系统全部封锁管理
  - 所有站点都向主站点提出封锁和解锁请求，由它去处理

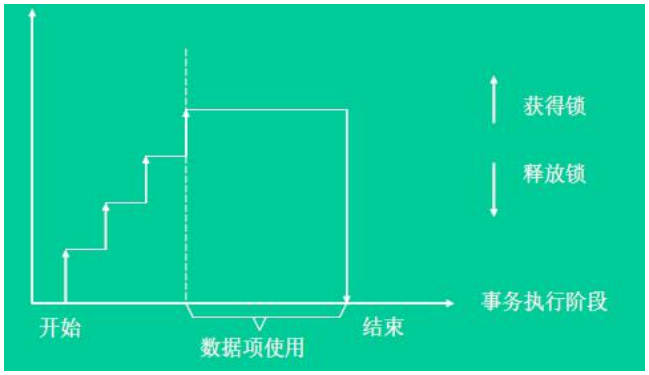
– 缺点有：
• 所有封锁请求都送往单个站点，容易由于超负荷造成“瓶颈”
• 主站点故障造成会使系统瘫痪，封锁消息都在这里，制约了系统可用性和可靠性
• 主副本封锁法
– 不指定主站点，指定数据项的主副本
– 事务对某个数据项进行操作时，先对其主副本进行封锁，再进行操作
– 主副本封锁，意味着所有的副本都被封锁
– 主副本按使用情况，尽量就近分布
– 可减少站点的负荷,使得各站点比较均衡
– 可减少传输量

2、2PL 协议（两阶段封锁协议）
（1）基本 2PL 协议
* 2PL 可以保证事务执行的可串行性
• 如果一个事务所有的封锁操作（读写）都在第一个解锁操作之前，那么它就遵守 2PL 协议
• 事务的执行中 Lock 的管理分成两个阶段
– 上升阶段(成长阶段)：获取 Lock 阶段
– 收缩阶段(衰退阶段）：释放 Lock 阶段
• 封锁点是指事务获得了它所要求的所有锁，并且还没有开始释放任何锁的时刻
• 如果允许锁的转换，锁的升级必须在成长阶段进行，锁的降级必须在锁的衰退阶段进行。
（2）保守 2PL
– 要求事务在开始执行之前就持有所有它要访问的数据项上的锁
– 事务要预先声明它的读集和写集
** 大多数 2PL 调度器实现的是严格 2PL(S2PL)
– 事务在提交或者撤销之前，绝对不释放任何一个写锁
– 事务结束时（提交或者撤销），同时释放所有的锁



（3）严酷 2PL

- 事务 T 在提交或撤销之前，不能释放任何一个锁（写锁或者读锁），因此它比严格 2PL 更容易实现



（4）保守 2PL 与严酷 2PL 之间的区别

- 前者，事务必须在开始之前封锁它所需要的所有数据项，因此，一旦事务开始就处在收缩阶段
- 后者,直到事务结束（提交或者撤销）后才开始解锁，因此，事务一直处于扩张阶段，直到结束

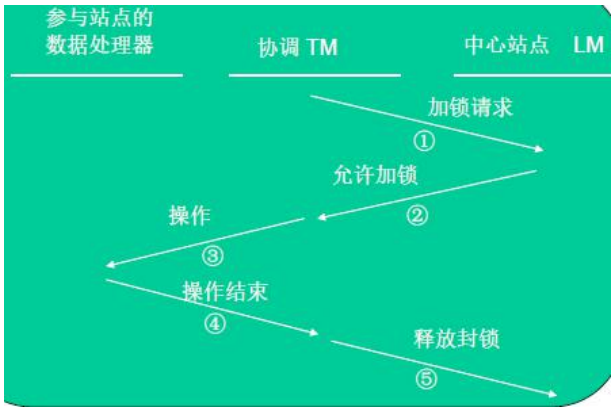
3、2PL 协议的实现方法

（1）集中式 2PL 的实现方法

2PL 很容易扩展到分布式 DBMS(无论复制或无复制 DDB)，其最简单的方法是选择一个站点(主站点)做 Lock 管理器，其他站点上的事务管理器都需要与该选出的站点 Lock 管理器通信，而不是与本站点 Lock 管理器通信。

* 相关术语

- 协调事务管理器(coordinating TM) ：事务原发站点
- 数据处理器(data processor，DP) ：其他参与站点
- 中心站点 LM：主站点锁管理器



** 中心站点不需要想数据处理器发送操作

（2）主副本 2PL 的实现方法（是主站点 2PL 的直接扩展）

- 选择一组站点做 Lock 管理器
- 每个 Lock 管理器管理一组数据(即每个数据选择一个站点作自己的 Lock 管理器)
- 事务管理器根据 Lock 申请的数据对象分别向这些数据的 LM 发出锁申请
- 必须先为每一个数据项确定一个主副本站点

（3）分布式 2PL 的实现方法

* 特点

- 每个站点都有 LM
- 无副本 DDB 上如同主副本 2PL
- 有冗余副本 DDB，等待一定数量副本加锁成功

* 与集中式区别

- 集中式中向中心站点封锁管理程序发送的信息，在分布式中发送给所有参与站点的封锁管理程序
- 分布式通过参与者的封锁管理程序
- 参与者的数据处理器向协调者的事务管理程序发送“操作结束”信息，协调者事务管理器向参与者事务管理器发送解锁指令

4、多粒度封锁与意向锁

（1）多粒度封锁

- 封锁的粒度不是单一的一种粒度，而是有多种粒度
- 可以定义多粒度树，根节点是整个数据库，叶节点表示最小的封锁粒度

（2）直接封锁

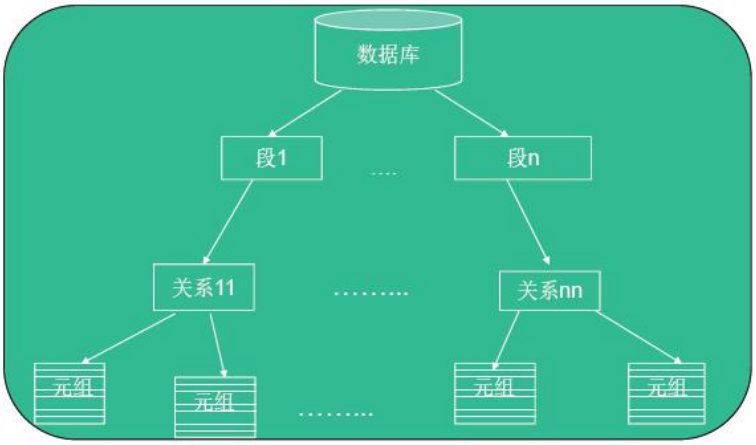
- 事务对要进行读/写的数据对象直接申请加锁

（3）分层封锁

- DB 中各数据对象从大到小存在一种层次关系，例如划分为 DB，段，关系，元

组，字段等

- 当封锁了外层数据对象时，蕴含着也同时封锁了它的所有内层数据对象
- 数据项的显式封锁和隐式封锁



多级粒度树

** 举例

假定事务 T1 要更新文件 f1 中的所有记录，T1 请求并获得了 f1 上的一个写锁

- 那么 f1 下面的页面和记录就获得了隐式写锁
- 如果这时候，事务 T2 想从 f1 中的某个页面中读某个记录，那么 T2 就要申请该记录上的读锁
- 但是要确认这个读锁和已经存在锁的相容性，确认的方法就是要遍历该树：从记录到页，到文件最后到数据库，如果在任意时刻，在这些项中的任意一个上存在冲突锁，那么对记录的封锁请求就被拒绝，T2 被阻止，要等待。

（4）意向锁

- 如果对一个节点加意向锁，则说明该节点的下层节点正在被封锁
- 对任一节点封锁时，必须先对它的上层节点加意向锁

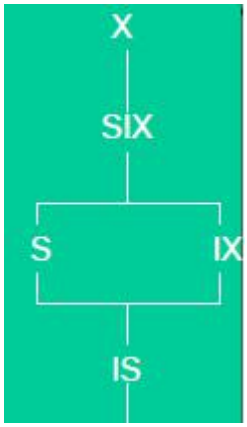
（5）意向锁的类型

- 意向共享锁(IS)：指示在其后代节点上将会请求共享锁
- 意向排它锁(IX)：指示在其后代节点上将会请求排他锁
- 共享意向排它锁(SIX)：指示当前节点处在共享方式的封锁中，但是在它的某些后代节点中将会请求排他锁。
- 例如：对某个表加 SIX 锁，则表示该事务要读整个表（加 S 锁），同时会更新个别元组（加 IX 锁）

（6）锁的相容矩阵

T_1 \ T_2	S	X	IS	IX	SIX	-
S	Y	N	Y	N	N	Y
X	N	N	N	N	N	Y
IS	Y	N	Y	Y	Y	Y
IX	N	N	Y	Y	N	Y
SIX	N	N	Y	N	N	Y
-	Y	Y	Y	Y	Y	Y

（7）锁的强度的偏序关系



（8）多粒度封锁协议的规则

- 必须遵守锁的相容性规则
- 必须首先封锁树的根节点，可以用任何一种方式的锁
- 只有当节点 N 的父节点已经被事务 T 以 IS 或 IX 方式封锁后，节点 N 才可以被 T 以 S 或者 IS 方式封锁
- 只有当节点 N 的父节点已经被事务 T 以 IX 或 SIX 方式封锁后，节点 N 才可以被 T 以 X,IX 或者 SIX 方式封锁
- 只有当事务 T 还没有释放任何节点时，T 才可以封锁一个节点
- 只有当事务 T 当前没有封锁节点 N 的任何子节点时，T 才可以为节点 N 解锁。

（9）总结

- 具有意向锁的多粒度加锁方法中，任意事务 T 要对一个数据对象加锁，必须先对它的上层节点加意向锁
- 申请封锁时应该按自上而下的次序进行
- 释放锁时则应该按自下而上的次序进行
- 具有意向锁的多粒度加锁方法提高了系统的并发度，减少了加锁和释放锁的开销
- 它已经在实际的 DBMS 系统中广泛应用，例如 Oracle 中

三、分布式数据库系统并发控制的时标技术

1、基于时标的并发控制方法

（1）基本概念

- 不通过互斥来支持串行性，而是通过在事务启动时赋给时标（时间戳）来实现
- 时标是用来唯一识别每个事务并允许排序的标识
- 如果 ts(T1) < ts(T2) … < ts(Tn), 则调度器产生的序是:T1,T2, ... Tn

（2）规则

- 如果 T1 的操作 O1(x)和 T2 的操作 O2(x)是冲突操作, 那么,O1 在 O2 之前执行, 当且仅当 ts(T1)<ts(T2)

（3）时标分配方法

- 全局时标

• 使用全局的单调递增的计数器
• 全局的计数器维护是个难题

- 局部时标

• 每个站点基于其本地计数器自治地指定一个时标
• 标识符由两部分组成：〈本地计数器值，站点标识符〉

- 站点标识符是次要的，主要是本地计数器值
- 可以使用站点系统时钟来代替计数器值

（4）时标法思想

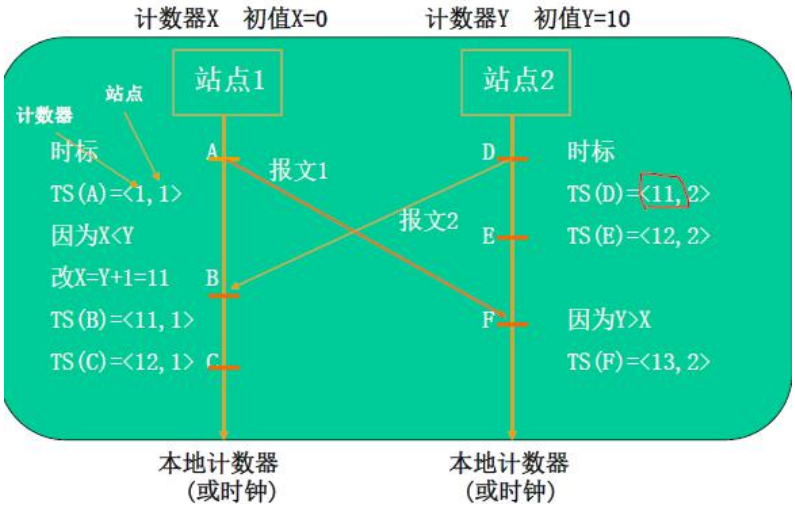
- 每个事务赋一个唯一的时标，事务的执行等效于按时标次序串行执行
- 如果发生冲突，是通过撤销并重新启动一个事务来解决的
- 事务重新启动时，则赋予新的时标
- 优点是没有死锁，不必设置锁
- 封锁和死锁检测引起的通信开销也避免了
- 但要求时标在全系统中是唯一的

（5）时标性质

唯一性；单调性

（6）全局唯一时间的形成与调整

- 每个站点设置一个计数器, 每发生一个事务, 计数器加一
- 发送报文时包含本地计数器值, 近似同步各站点计数器



2、基本时标法

（1）规则

- 每个事务在本站点开始时赋予一个全局唯一时标
- 在事务结束前，不对数据库进行物理更新
- 事务的每个读操作或写操作都具有该事务的时标
- 对每个数据项 x，记下写和读操作的最大时标，记为 WTM(x)和 RTM(x)
- 如果事务被重新启动，则被赋予新的时标

（2）执行过程

- 令 read_TS 是事务对 x 进行读操作时的时标

• 若 read_TS<WTM(x)，则拒绝该操作，事务重新启动

• 否则，执行，令 RTM(x)=max{RTM(x), read_TS}

- 令 write_TS 是事务对 x 进行写操作时的时标

• 若 write_TS < RTM(x) 或 write_TS < WTM(x) ,则拒绝该操作，事务重新启动

• 否则，执行，令 WTM(x) = max{WTM(x), write_TS}

（3）缺点

重启动多

3、保守时标法

（1）基本思想

- 一种消除重启动的方法
- 通过缓冲年轻的操作，直至年长的操作执行完成，因此操作不会被拒绝，事务也绝不被重启动

（2）规则

- 每个事务只在一个站点执行，它不能激活远程的程序,但是可以向远程站点发读/写请求

– 站点 i 接收到来自不同站点 j 的读/写请求必须按时标顺序，即每个站点必须按时标顺序发送读/写数据请求，在传输中也不会改变这个顺序

– 每个站点都为其它站点发来的读/写操作开辟一个缓冲区，分别保存接收到的读/写申请

– 假定某个站点 k 上,各个缓冲区队列都已不为空，即每个站点都已向它至少发送了一个读和一个写操作，就停止接收，处理在缓冲区中的操作

– 假定站点 i 至少有一个缓冲的读和缓冲的写来自网中其它站点，根据规则 2, Site i 知道没有年老的请求来自其它 Site(因为按序接收，所以不可能有比此更年老的请求到来，年老的比年轻的先到)

** 举例：已知站点 i 的缓冲区队列中有来自所有站点的读/写请求如下所示:

站点1	站点2	站点3	……	站点n
R11	R21	R31		Rn1
R12	R22	R32		
R13	R23			
	R24			
W11	W21	W31	……	Wn1
	W22	W32	…	Wn2
	W23			

执行步骤:

(1) 设 RT=min(Rij), WT= min(Wij)

(2) 按下法处理缓冲区中的 Rij 和 Wij

a. 若队列中有(Rij) <= WT 的 Rij，则顺序执行这些 Rij，执行完删掉

b. 若队列中有(Wij) <= RT 的 Wij，则顺序执行这些 Wij，执行完删掉

(3) 修改 RT=min(Rij), WT=min(Wij) ，此时的 Rij 和 Wij 是队列中剩余的

(4) 重复上述(2)和(3)，直到没有满足条件的操作，或者:

a. 若某个或某些 R 队列为空时, RT=0;

b. 若某个或某些 W 队列为空时, WT=0

（3）存在问题和解决方法

– 如果一个站点从来不向某个站点发送操作的话，那么执行过程中的假定就不符合，操作就无法进行。解决办法是，周期性的发送带有时标的空操作

– 此方法要求网络上所有站点都连通，这在大系统中很难办到。为避免不必要的通信，可对无读写操作请求的站点，发送一个时标很大的空操作

– 此方法过分保守，一律按照时序来进行，其中包括了不冲突的操作

四、分布式数据库系统并发控制的多版本技术

1、多版本概念和思想

（1）基本思想

- 保存已更新数据项的旧值，维护一个数据项的多个版本
- 通过读取数据项的较老版本来维护可串行性，使得系统可以接受在其他技术中被拒绝的一些读操作
- 写数据项时，写入一个新版本，老版本依然保存

（2）缺点

需要更多的存储来维持数据库数据项的多个版本

（3）分类

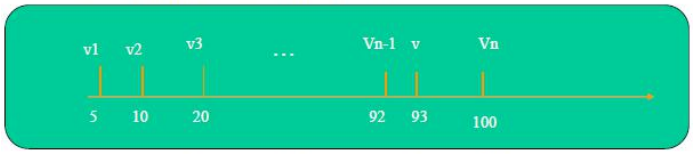
- 基于时标排序
- 基于两阶段封锁

2、基于时标的多版本技术

- 数据项 X 的多版本
  - X1, X2, X3,…, Xk
- 系统保存的值
  - Xi 的值
  - 两种时标
- Read_TS(Xi): 读时标，成功读取版本 Xi 的事务的时标，最大的一个
- Write_TS(Xi): 写时标，写入版本 Xi 的值的事务的时标
- 多版本规则
  - 如果事务 T 发布一个 write_item(X)操作，并且 X 的版本 Xi 具有 X 所有版本中最高的 write_TS(Xi),同时 write_TS(Xi)<=TS(T)且 read_TS(Xi)>TS(T)，那么撤销并回滚 T；否则创建 X 的一个新版本，并且令 read_TS(Xi)= write_TS(Xi)= TS(T)
  - 事务 T 发布一个 read_item(X)操作，如果 write_TS(Xi)<=TS(T)并且 X 的版本 Xi 具有 X 所有版本中最高的 write_TS(Xi),则把 Xi 的值返回给事务 T,并且将 read_TS(Xi)的值置为 TS(T)和当前 read_TS(Xi)中较大的一个

若读 TS(Ri)=95, 则读 <92, Vn-1> 的值

若写TS(Wk)=93, 则出现了矛盾



于是要拒绝TS(Wk), 否则 TS(Ri)=95读的就是Vn-1, 而不是v的值, 但是按规定 TS(Ri)=95应该读的是v值

3、采用验证锁的多版本两阶段封锁

- 三种锁方式
  - 读，写，验证
- 四种锁状态
  - 读封锁（read_locked）
  - 写封锁（write_locked）
  - 验证封锁（certify_locked，写提交前验证没有其他读）
  - 未封锁（unlocked）
- 锁相容性
  - 标准模式锁相容性（写锁和读锁）
  - 验证模式锁相容性（写锁、读锁和验证锁）

	读	写
读	是	否
写	否	否

(a) 读/写封锁模式的相容性表

	读	写	验证
读	是	是	否
写	是	否	否
验证	否	否	否

(b) 读/写/验证封锁模式的相容性表

- 多版本 2PL 的思想
  - 当只有一个单独的事务 T 持有数据项上的写锁时，允许其他事务 T’读该项 X，这是通过给予每个项 X 的两个版本来实现的
- 一个版本 X 是由一个已提交的事务写入的
- 另一个版本 X’是每个事务 T 获得该数据项上写锁时创建的、
  - 当事务 T 持有这个写锁时，其他事务可以继续读 X 的已提交版本
  - 事务 T 可以写 X’的值，而不影响 X 已提交版本的值
  - 但是，一旦 T 准备提交，它必须在能够提交之前，得到持有写锁的数据项上的验证锁（要等所有读锁释放之后）
  - 一旦获得验证锁，数据项的已提交版本 X 被置为版本 X’的值，版本 X’被丢弃，验证锁被释放。

五、分布式数据库系统并发控制的乐观方法

1、基本思想和假设

（1）基本思想

对于冲突操作不像悲观方法那样采取挂起或拒绝的方法，而是让一个事务执行直到完成

（2）基于如下假设

- 冲突的事务是少数（查询为主的系统中，冲突少于 5%）
- 大多数事务可以不受干扰地执行完毕

2、执行阶段划分

悲观法事务执行的各个阶段

乐观法事务执行的各个阶段