
第十章

数据库管理

第十章 数据库管理

10.1 数据库管理概述

10.2 数据库管理的内容

10.3 数据库管理员DBA

10.4 数据库性能配置和优化

10.1 数据库管理概述

- 数据库是一种共享的复杂的数据体，在数据库设计完成后，经过数据库建立、数据加载之后投入实际运行，并在运行过程中进行监控和维护。上述的这些管理维护工作被称为数据库管理。
- 实施管理工作的人则被称为数据库管理员（**database administrator**，简称**DBA**）。

10.1 数据库管理概述

➤ 在数据库系统对数据库的管理有两种方式:

用软件对数据作管理的**DBMS**

用**DBA**对数据作管理的数据库管理

10.1 数据库管理概述

- **DBA对数据作管理的数据库管理**
 - 数据库的建立
 - 数据库的调整
 - 数据库的重组
 - 数据库的重构
 - 数据库的安全性控制与完整性控制
 - 数据库的并发控制
 - 数据库的故障恢复
 - 数据库的监控

10.1 数据库管理概述

10.2 数据库管理的内容

10.3 数据库管理员DBA

10.4 数据库性能配置和优化

10.2 数据库管理的内容

1. 数据库的建立

- 在完成数据库设计之后，要得到可以实际运行的数据库，还需要经过一个数据库的建立过程。
- 数据库的建立包括：
 - 数据库运行环境的设置
 - 数据模式的建立
 - 会话环境的建立
 - 数据加载

10.2 数据库管理的内容

数据库运行环境的设置

- 在建立数据库之前，应首先了解所选用的**DBMS**及其对运行环境的要求，熟悉**DBMS**的各种运行参数的含义、缺省值及其对系统的影响，确定新系统中各个参数的取值。
- 不同的**DBMS**有不同的参数，一般有下列三种类型：
 - 有关内外存分配的参数，如数据文件的大小、数据块的大小、最大文件数、缓冲区的大小等；
 - 有关**DBMS**运行参数的设置，如可同时连接的用户数、可同时打开的文件和游标数量、日志缓冲区的大小等；
 - 有关数据库的故障恢复和审计跟踪的参数，如审计功能的开启/关闭参数、系统日志的设置等。

10.2 数据库管理的内容

数据模式建立

- **DBA**利用**DBMS**中的**DDL**语言以及相应的运行工具来定义和执行数据库的数据建模脚本。
- 建模脚本的任务：
 - 建立数据模式，定义数据库名；
 - 定义分区并申请初始存储空间和各种系统资源；
 - 定义表及相应属性；
 - 定义表中的主键、外键及其它数据完整性约束；
 - 定义索引、集簇等性能优化措施；
 - 定义各个授权用户在表上的访问权限；
 - 通过定义视图来定义用户外模式；
 - 为进行数据交换还需设置会话环境参数

10.2 数据库管理的内容

会话环境的建立

- 在网络上运行的数据库在建立模式后尚需建立会话环境，如会话语言、时间的设定、会话数据客体模式设定以及最终会话标识符的设定。

数据加载

- 数据加载的准备工作，包含数据采集、整理及校验等工作
- 一般**DBMS**都提供加载程序以供数据加载之用。
- 有些数据可以从其它数据体（如其它文件、数据库或网络上的其它结点）获取，此时必须选用相应的转换程序以实现加载。

10.2 数据库管理的内容

2. 数据库调整

- 在数据库建立并经一段时间运行后，往往会产生一些不适应的情况，如：
 - 最初建立的数据模式不能完全满足用户应用的数据需求；
 - 由于增加了新的应用而带来新的数据需求；
 - 随着数据库中数据的不断增加，系统的性能也有可能下降而无法满足应用程序在性能方面的需求。
- 这就需要对数据库进行调整。调整内容主要包括：

10.2 数据库管理的内容

(1) 调整关系模式与视图使之更能适应用户的数据需求

- 如果是因数据项的缺失而引起的数据库调整，那么可以修改关系表的属性定义。
- 如果是为改善某些应用的性能，可以采取逆规范化、关系分割以及建立快照等措施。
- 如果是用户的数据需求发生了变化，那么就需要修改外模式，定义新的视图或修改原来的视图。

10.2 数据库管理的内容

- **逆规范化**：是将若干个子关系模式合并为一个关系模式，减少应用访问过程中的关系连接次数，从而改善应用的执行性能。
- **关系分割**：可以使用户的应用只需要访问某个子关系中的属性和元组，从而减少被访问关系中的数据量，达到提高数据访问性能的目的。关系分割又分为水平分割和垂直分割：
 - 水平分割是将一个关系中的元组集合划分为互不相交的若干个子集，为每个子集建立一个独立的子模式。
 - 垂直分割是将一个关系中的属性划分为若干个属性子集，每个属性子集单独构成一个子模式，垂直分割需要满足无损联接性。
- **快照**：通过查询结果的实例化存储来避免对于查询结果的重复使用而带来的查询开销。

10.2 数据库管理的内容

(2) 调整索引与集簇使数据库性能与效率更佳

- 集簇和索引的设计是数据库物理设计的主要内容，也是数据库调整的任务之一。
- 集簇和索引的设计通常是针对用户的核心应用及其数据访问方式来进行的，随着数据库中数据量的变化以及各个用户应用的重要性程度的变化，可能需要调整原来的集簇和索引的设计方案，撤消原来的一些集簇或索引，建立一些新的集簇和索引。

(3) 调整分区、调整数据库缓冲区大小以及调整并发度，使数据库物理性能更好

10.2 数据库管理的内容

- 通过上述数据库运行环境或参数的调整以达到提高系统性能的目的。如调整数据在不同磁盘驱动器上的分布情况，调整数据库缓冲区的大小和系统的最大并发用户数等。

10.2 数据库管理的内容

3. 数据库重组

- 数据库在经过一定时间运行后，其性能会逐步下降，下降的原因主要是由于不断的修改、删除与插入所造成的。
- 数据库重组：对数据库进行重新整理，重新调整存贮空间。
 - 一般数据库重组往往是先作数据卸载（**unload**），然后再重新加载（**reload**），即将数据库的数据先行卸载到其它存储区域中，然后按照模式的定义重新加载到指定空间，从而达到数据重组的目的。
 - 目前一般**RDBMS**都提供一定手段，以实现数据重组功能。

10.2 数据库管理的内容

4. 数据库重构

- 数据库重构即是数据库模式的重新构造。在数据库系统使用过程中由于应用环境改变而需求改变时，要求对数据库的模式作重大的变动，这称数据库重构。
 - 数据库的重构与大量的应用程序紧密相关，这主要是数据库中结构、命名、规则的改变而影响应用程序的调用，因此在重构时最大的原则是尽量减少应用程序的改动。
 - 一种比较有效的办法是采用视图的方法，将模式修改部分尽可能对原有应用程序屏蔽。

10.2 数据库管理的内容

5. 数据库的安全性控制

➤ 数据的安全性控制包括：

- 外部环境的安全：主要是通过行政手段，并建立一定的规章制度以确保数据库运行环境的安全。
- 数据库管理系统内部的安全
- 其它计算机系统内部的安全：主要是指操作系统的安全以及病毒防护安全。

10.2 数据库管理的内容

6. 数据完整性控制

➤ 完整性控制主要包括如下内容：

(1) 通过完整性约束检查的功能以保证数据的正确性。

- ❖ 首先，系统具有自动检验实体完整性与参照完整性能力；
- ❖ 其次，通过设置域约束、表约束及断言以建立数据的语义约束；
- ❖ 最后可以通过设置触发器以建立较为复杂、有效的完整性约束机制。

(2) 建立必要的规章制度进行数据的及时、正确的采集及校验。

- ❖ 数据库中数据大都采自外界，对这些采集来的数据的正确性保证是必须建立必要的规章制度及校验制度，从而将错误消灭在数据录入前。

10.2 数据库管理的内容

7. 数据库的并发控制

➤ 性能的调整

- 当并发控制执行时，由于环境与应用的改变经常会出现并发控制性能的不稳定，此时须即时调整并发度以取得较高的执行效率。

➤ 死锁、活锁的解除

- 并发控制执行时经常会产生活锁与死锁现象，此时须采取相应措施调整事务的执行以解除事务的活锁与死锁，使并发控制得以继续进行。

10.2 数据库管理的内容

8. 数据库的故障恢复

- 数据库中数据可能会遭受破坏，这种破坏可来自外部和内部两个方面，数据在一旦遭受破坏后能及时进行恢复是数据库管理的基本任务。
- **DBMS**一般都提供故障恢复的有关手段，并由**DBA**负责执行故障恢复功能。
- 常用故障恢复的方法：
 - 定期的转储
 - 日志
 - **DBA**利用日志、拷贝及**REDO**、**UNDO**等手段，根据不同的故障类型（小型、中型、大型）及类别进行故障恢复。

10.2 数据库管理的内容

9. 数据库的监控

- **DBA**使用监控工具随时观察数据库的动态变化，这种行为称为数据库监控。
- 数据库的监控是数据管理的基本任务，也是**DBA**的日常工作之一。

10.1 数据库管理概述

10.2 数据库管理的内容

10.3 数据库管理员DBA

10.4 数据库性能配置和优化

10.3 数据库管理员DBA

- **DBA**是管理数据库的核心人物，他一般由若干个人员组成，是数据库的监护人，也是数据库与用户间的联系人。
- **DBA**具有最高级别的特权，他对数据库系统应有足够的了解与熟悉，一个数据库能否正常、成功的运行，**DBA**是关键。
- **DBA**除了完成数据库管理的工作外，还需要完成相关的行政管理工作以及参与数据库设计的部分工作。

10.3 数据库管理员DBA

➤ DBA的具体任务:

- 1) 参与数据库设计的各个阶段的工作，对数据库有足够的了解。
- 2) 负责数据库的建立、调整、重组与重构。
- 3) 维护数据库中数据的安全性。
- 4) 维护数据库中数据的完整性以及对并发控制作管理。
- 5) 负责数据库的故障恢复及制订灾难恢复计划。
- 6) 对数据库作监控，及时处理数据库运行中的突发事件并对其性能作调整。

10.3 数据库管理员DBA

7) 帮助与指导数据库用户

与用户保持联系，了解用户需求，倾听用户反映，帮助他们解决有关技术问题，编写技术文档，指导用户正确使用数据库。

8) 制定必要的规章制度，并组织实施。

为便于使用管理数据库，需要制订必要的规章制度，如数据库使用规定，数据库安全操作规定，数据库值班记录要求等，同时还要组织、检查及实施这些规定。

10.1 数据库管理概述

10.2 数据库管理的内容

10.3 数据库管理员DBA

10.4 数据库性能配置和优化

10.4 性能配置和优化

- 用户可以配置一些参数以提高性能，其中有一些参数会显著影响实例和数据库运行的性能
- 性能是计算机系统特定工作负荷下的表现，可通过一个或多个系统的响应时间、处理能力和可用性来测量衡量。性能受到以下因素影响
 - 可用的资源；
 - 如何充分利用这些资源

10.4 性能配置和优化

- 如果要改善系统的成本效益比例，应该优化性能，具体情形如下：
 - 在不增加处理成本的情况下，处理一项更大的、要求更高的工作
 - 在不增加处理成本的情况下，获得更快的系统响应时间或者更高的处理能力
 - 在不对用户服务产生负面影响的情况下，降低处理成本

10.4 性能配置和优化

➤ 优化原则

- 牢记边际收益递减规律
- 不要为优化而优化
- 考虑整个系统
- 一次更改一个参数
- 按级别测量和重新配置
- 检查硬件和软件的问题
- 在升级硬件前明确问题
- 在开始优化前执行备份过程

10.4 性能配置和优化

➤ 性能改进过程

- 建立性能指标
- 定义性能目标
- 制定性能监视方案
- 实施该方案
- 分析衡量因素，以确定是否满足了目标要求
- 确定系统中的主要约束
- 决定在哪些方面可进行性能改进以及哪些资源可承受附加的负荷
- 调整系统配置
- 根据结果，继续对系统进行下一轮监视
- 对定期监视或在对系统、工作负荷做重大更改时，用户都可按上述过程进行

10.4 性能配置和优化

➤ 可对系统进行多大程度的优化

- 系统可能不需要任何优化也可以运行的很好，但可能没有发挥它的潜能
- 当系统遇到性能瓶颈时，优化可能发挥作用
- 如果已接近性能极限，同时又将系统上的用户数增加了大约**10%**，那么响应时间可能远不止增加**10%**

➤ 一种不太正式的方法

- 如果没有足够的时间来设定性能目标并通过一种完备的方式来监控和优化，可听听用户的意见以改善性能，了解他们是否有与性能相关的问题

10.4 调整配置参数

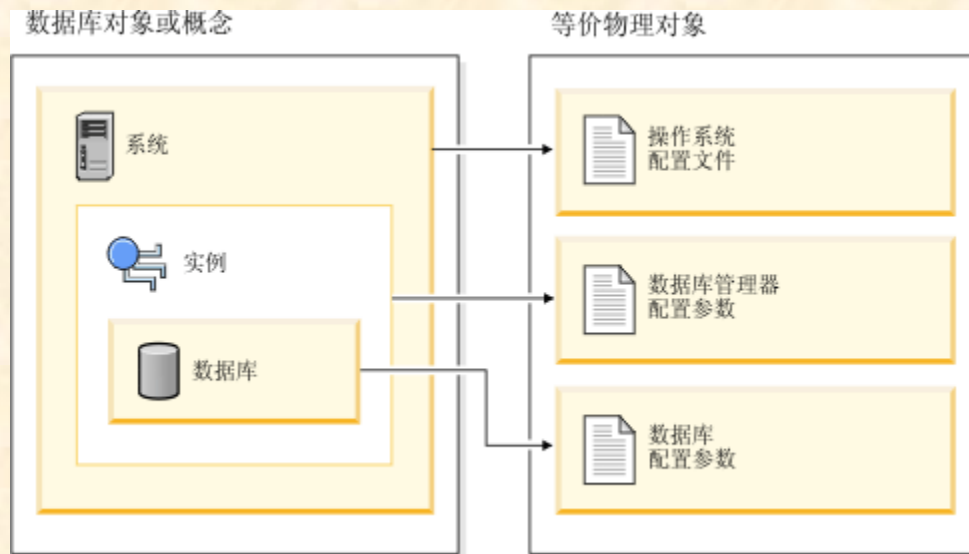
- 数据库管理程序根据默认的参数值分配磁盘空间和内存，可能足以满足用户的需要。但在某些情况下，使用这些默认值可能无法实现最佳性能
 - 默认值适用于内存相对较小且专门用作数据库服务器的机器
 - 如果用户的环境存在以下情况，可能需要修改这些参数
 - ❖ 大型数据库
 - ❖ 大量连接
 - ❖ 对特定应用程序有高性能需求
 - ❖ 唯一的查询或事务负荷或类型
 - ❖ 不同的机器配置或用途

10.4 调整配置参数

- 数据库管理器配置参数存储在名为 **db2sysm** 的文件中。此文件是在创建数据库管理器的实例时创建的。在 **Linux** 和 **UNIX** 环境中，可以在数据库管理器的实例的 **sqlib** 子目录中找到此文件
- 在 **Windows** 中，此文件的缺省位置随 **Windows** 操作系统系列的版本不同而有所变化；要验证 **Windows** 上的缺省目录，请使用 **DB2SET DB2INSTPROF** 命令来检查 **DB2INSTPROF** 注册表变量的设置。您还可以通过更改 **DB2INSTPROF** 注册表变量来更改缺省实例目录。如果设置了 **DB2INSTPROF** 变量，那么文件将在 **DB2INSTPROF** 变量所指定的目录的 **instance** 子目录中

10.4 调整配置参数

- 不能直接编辑db2sysm文件，只能使用提供的API或通过调用该API的工具来更改或查看它。
- 数据库对象与配置文件之间的关系



10.4 调整配置参数

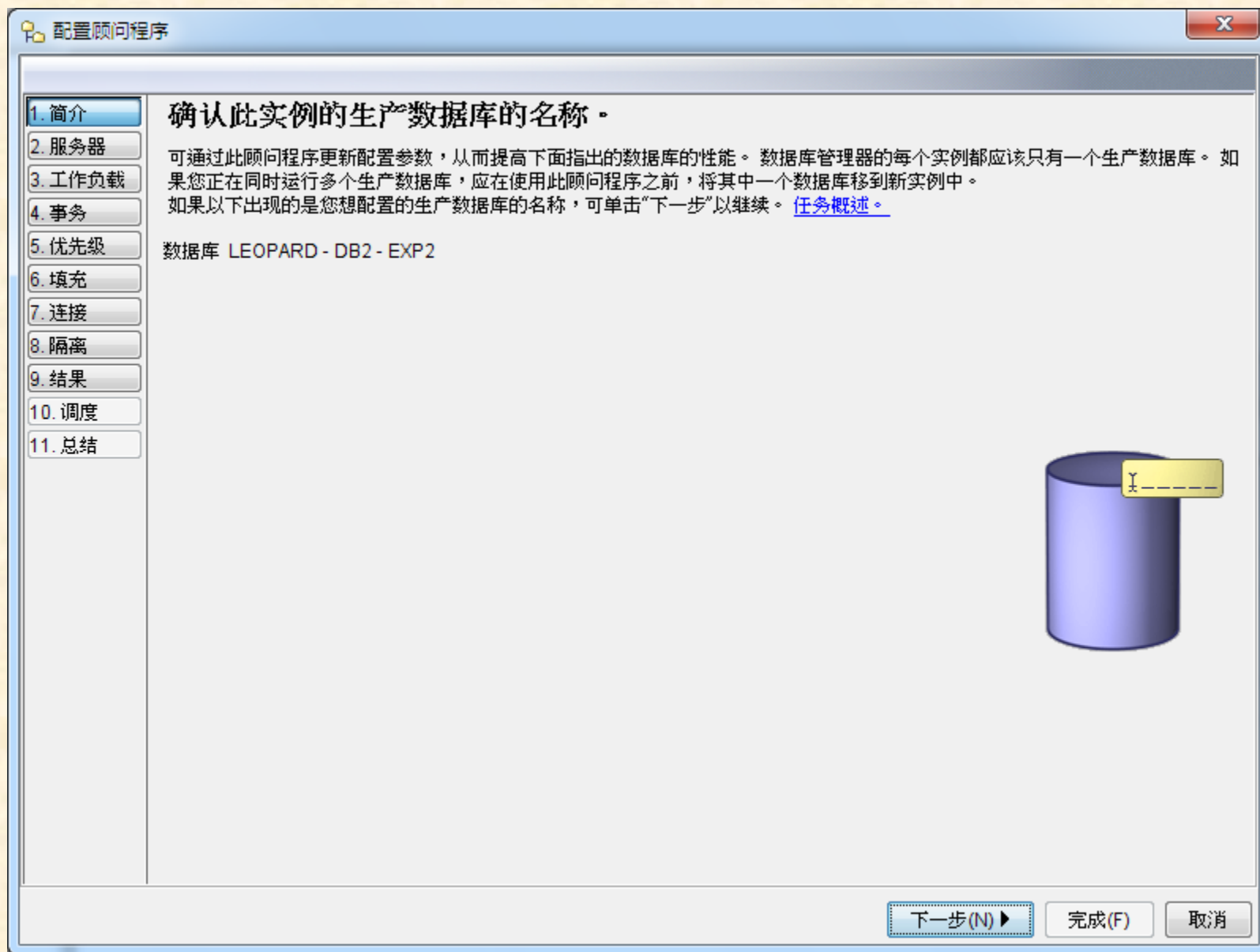
➤ 数据库参数

- 使用控制中心，提供了“配置顾问程序”来改变配置参数的值
 - ❖ 此向导根据用户对一组问题（如对数据库运行的事务的工作负荷和类型）提供的回答来为参数生成值
- 为了查看数据库配置文件中的配置参数的当前值，可以使用：

GET DATABASE CONFIGURATION

或 **get db cfg**

10.4 调整配置参数



10.4 调整配置参数

- 为了更新数据库管理器配置文件中的单个配置参数的取值，可以使用

UPDATE DATABASE CONFIGURATION

USING <parameter name> <new value>,

<next parameter name> <next value>, ...

- 为了重置数据库配置文件中的所有配置参数值，可以使用

RESET DATABASE CONFIGURATION

10.4 缓冲池对性能的影响

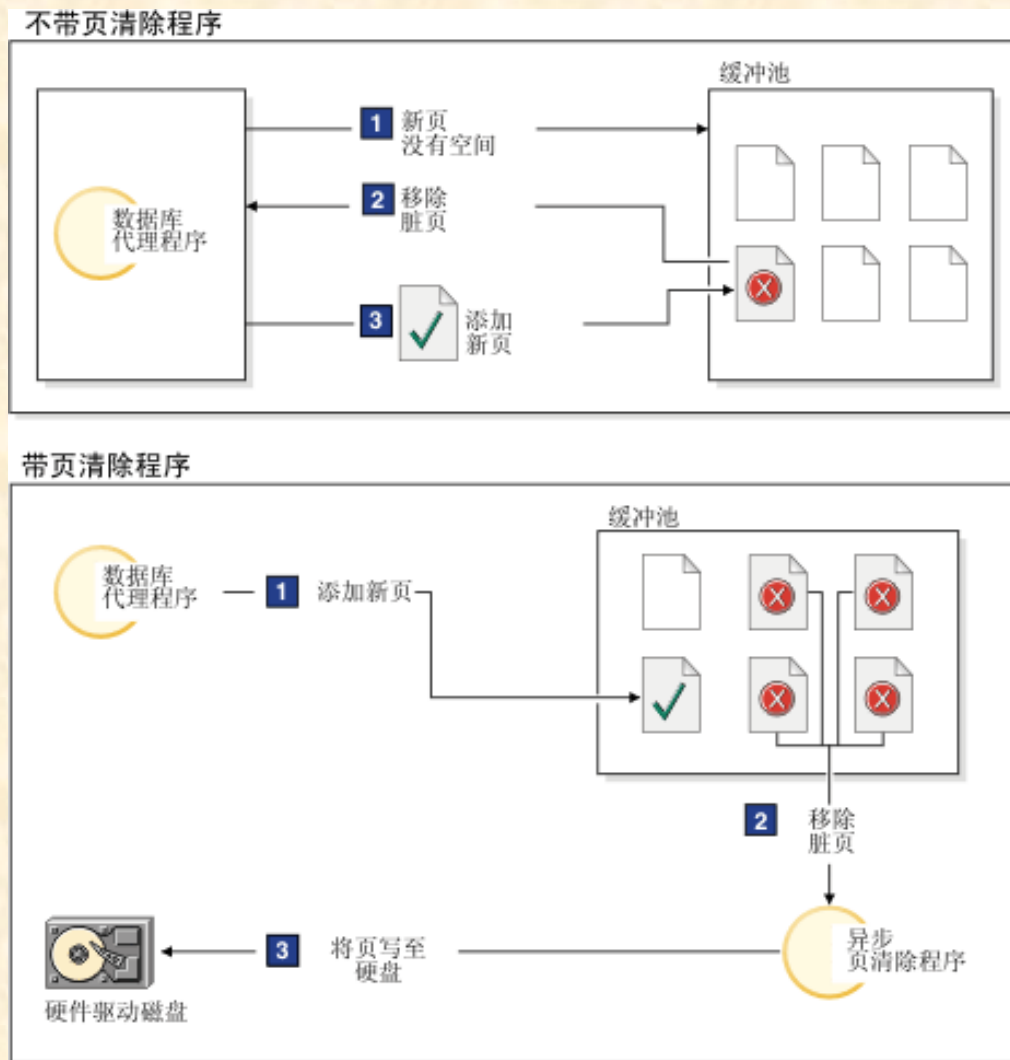
- 缓冲池是进行数据库页临时读取和更新的内存区域
- 从内存中访问数据的速度远比从磁盘上访问数据快
 - 所以数据库管理器从磁盘读取和写入数据的次数越少，数据库管理器和数据库的整体性能就越好
 - 从缓冲池中写入和读取数据的速度将会对数据库管理器处理数据查询的速度产生影响

10.4 缓冲池对性能的影响

- 在数据库关闭或者数据所占内存空间需要被其他数据页使用之前，数据页会一直保存在缓冲池中
- 数据库代理能够负责将旧数据或者无用数据移出缓冲池，并用新的、有用的数据替代它们
- 在缓冲池中实际上有两种类型的数据页
 - 使用中的数据页
 - ❖ 正在被读取或者更新的数据
 - 脏页面
 - ❖ 已经被修改但是还没有写回到磁盘上的数据页

10.4 缓冲池对性能的影响

➤ 缓冲区处理脏页的过程



10.4 缓冲池对性能的影响

- 所有缓冲池的大小都将对数据库的性能产生很大的影响。为了确保不会发生大量数据页交换，需要考虑以下因素
 - 计算机上安装内存的大小
 - 在同一台计算机上，可能与数据库管理器同时运行的应用程序所需的内存
- 当计算机内存不足以保存所有受访问页面的时候，就会出现数据页的交换
 - 导致部分数据页写入或交换到磁盘存储器的临时存储中，为其他数据页提供内存空间
 - 当需要访问临时存储在磁盘存储器上的页面时，就会将这些数据重新交换到主内存中

10.4 缓冲池对性能的影响

- 当计算机上出现以下环境时，可能需要将系统中**75%**以上的内存设置为数据库缓冲池
 - 多用户
 - 计算机只是当做数据库服务器使用
 - 对相同数据页或索引页的大量重复性访问
 - 计算机中只有一个数据库
- 可以使用数据库系统监控器计算缓冲池的命中率，从而帮助用户调整缓冲池

10.4 缓冲池对性能的影响

➤ CREATE BUFFERPOOL 语句

❖ **CREATE BUFFERPOOL <bufferpool name>
SIZE <number of pages> PAGESIZE 4096**

- **Bufferpool name:** 系统编目中标识缓冲池的单一名称
- **Number of pages:** 分配给缓冲池的页面数目。用户可以使用值-1，表示缓冲池的大小应该由数据库配置参数**buffpage**的值决定
- **PAGESIZE:** 定义了缓冲池使用的页面的大小。有效的取值为**4096, 8192, 16384, 32768**
 - ❖ 使用**K**后缀时，有效取值是**4, 8, 16, 32**

10.4 缓冲池对性能的影响

➤ ALTER BUFFERPOOL语句

- 修改缓冲池的大小
- 将扩展存储开启或关闭
- 在新的节点组中增加缓冲池的定义

➤ DROP BUFFERPOOL语句

- 用户缓冲池的删除语句
- 删除前要确保没有表空间被分配到这个缓冲池中，而且用户无法删除**IBMDEFAULTBP**缓冲池
- 在数据库停止前，缓冲池的存储都不会被释放

10.4 组织数据库中的数据

- 使用索引的**SQL**语句的性能会在许多更新、删除或者插入操作后下降，这些活动使得索引中的信息更加分散
- 经过一段时间之后数据中间位置就会受到其他数据的填充，所以逻辑上相连的数据就不能够在物理上靠近相关联的数据放置
- 当随后数据库管理器需要访问数据时，就会需要额外的读取操作
- 应该对表和索引数据进行周期性的重新组织，以确保性能不会受损太多

10.4 组织数据库中的数据

- 重新对表进行组织需要花费时间
- 用户也应该考虑更新数据库统计，这个操作需要的时间较少
- 在最新的数据库和表统计的帮助下，优化器就能知道最有效的访问数据的路径
 - 数据维护过程从**RUNSTATS**和**REORGCHK**程序开始
 - 在执行了**RUNSTATS**后，**REORGCHK**程序检查收集来的统计数据，应用**8**个公式，然后给出建议是否需要**REORG**

10.4 组织数据库中的数据

➤ 分析数据的物理结构

- REORGCHK命令会计算数据库上的统计，以判断表是否需要重新组织

REORGCHK ON TABLE <table name>

- REORGCHK命令会计算通过8个不同公式获得的统计，以判断性能是否受到了损害，或者是否可以通过重新组织表对性能进行改善。这些规则生成显示分配空间和被使用的空间之间关系的建议

❖ 3个规则使用在表上

❖ 5个用在索引上

10.4 组织数据库中的数据

命令编辑器 4 - DB2COPY1

命令编辑器(C) 所选项(S) 编辑(E) 视图(V) 工具(T) 帮助(H)

命令 查询结果 访问方案

目标 EXP2 添加(A)...

REORGCHK ON TABLE WHU.DEPT

F3: 100 * (需要页数/总页数) > 80

SCHEMA.NAME	CARD	OV	NP	FP	ACTBLK	TSIZE	F1	F2	F3	REORG
表: WHU.DEPT										
	3	0	1	1	-	75	0	-	100	--

索引统计信息:

F4: CLUSTERRATIO 或正常化的 CLUSTERFACTOR > 80
F5: 100 * (叶子页的已用空间/非空叶子页的可用空间) > MIN(50, (100 - PCTFREE))
F6: (100 - PCTFREE) * (在一个较小层索引中的可用空间数量/所有键所需的数量) < 100
F7: 100 * (伪删除的 RID 数/RID 总数) < 20
F8: 100 * (伪空叶子页数/叶子页总数) < 20

SCHEMA.NAME	INDCARD	LEAF	ELEAF	LVLS	NDEL	KEYS	LEAF_REC_SIZE	NLEAF_REC_SIZE	LEAF_PAGE_OVERHEAD	NLEAF_PAGE_OVERHEAD	PCT_PAGES_SAVED	F4	F5	F6	F7	F8	REORG
表: WHU.DEPT																	
索引: WHU.DEPT_PK_ID																	
	3	1	0	1	0	3	4	4	710	710	0	100	-	-	0	0	----

CLUSTERRATIO 或正常化的 CLUSTERFACTOR (F4) 将指示索引需要

语句终止字符:

显示命令结果。SQL 结果在“查询结果”页上返回。存取方案以图形方式显示在“存取方案”页上。

10.4 组织数据库中的数据

➤ 规则F1、F2和F3提供重组的方针

- **F1:** 处理溢出行的数量，在有**5%**或者更多的行溢出时会建议对表进行重组
 - ❖ 在一个新的字段被加入到一个表或者是一个可变长度的值增加了它的尺寸时可能发生溢出
- **F2:** 处理空闲和未使用的空间。如果表大小小于或者等于分配给这个表的总空间的**70%**的时候，建议对表进行重组
- **F3:** 处理空白页面。当表里多于**20%**的页面是空闲的时候，建议对表进行重组。如果一个页面中没有数据行，则被认为是空白的
- **REORG**字段里显示一个星号，就表示需要进行表重组

10.4 组织数据库中的数据

- 规则**F4、F5、F6、F7和F8**提供了对索引重组的建议
 - **F4**: 以同样的物理顺序存储的资料行（如索引）的百分比
 - **F5**: 计算为索引项保留的空间。分配给索引的空间**50%**以下应该是空的
 - **F6**: 调节索引页的用法。可以处理的索引页的数量大于总量的**90%**
 - **F7**: 计算删除的行的数量。应该小于总行数的**20%**
 - **F8**: 计算空页面数的比重。应小于总页面数的**20%**
 - **REORG**字段里显示一个星号，就表示需要进行表重组

10.4 组织数据库中的数据

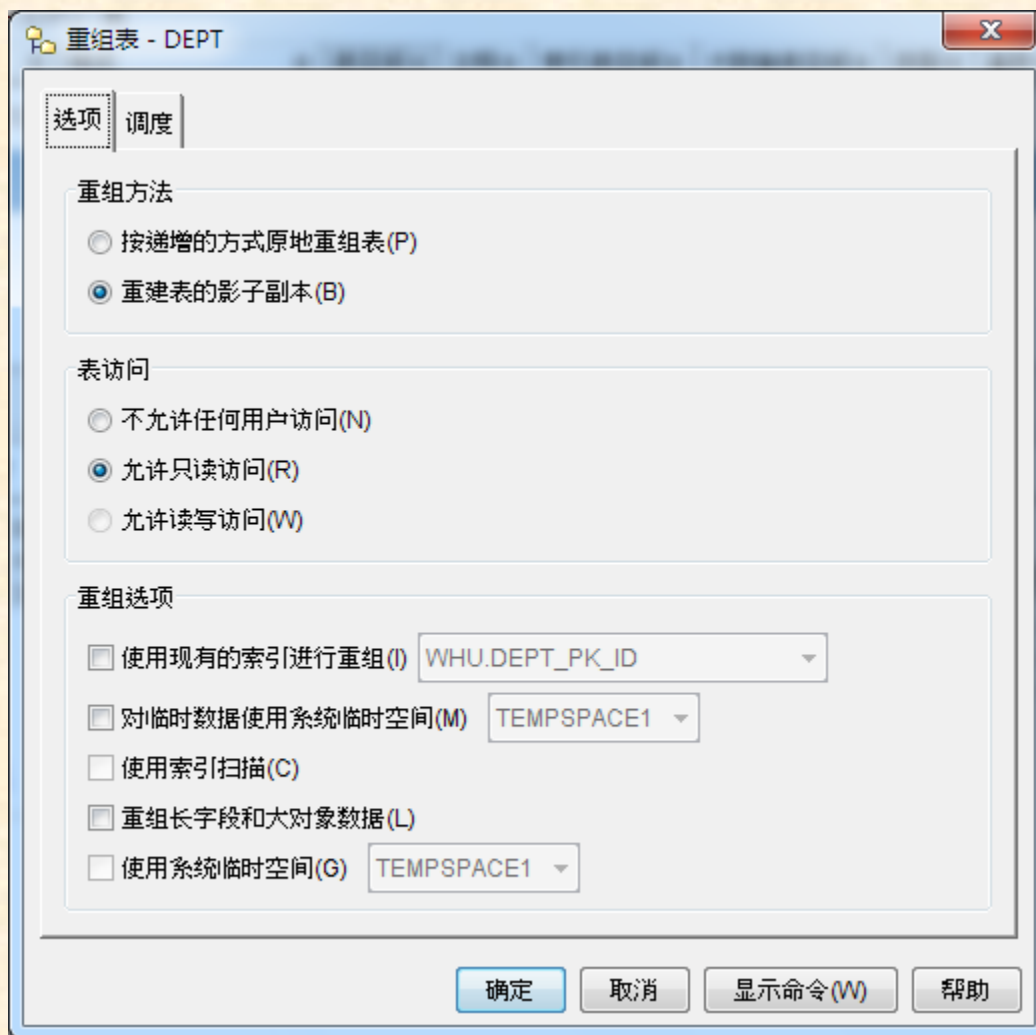
➤ 表重组REORG

- 在使用**REORGCHK**程序之后，可能发现表需要物理重组
- **REORG**程序将删除所有没有使用的空间并且把表和索引数据写入相互临近的页面中
- 在索引的帮助下，它也可以用来把资料行放置成同一个物理顺序（如同索引一样）

REORG TABLE <table name>

10.4 组织数据库中的数据

➤ 从控制中心使用REORG程序



重组表 - DEPT

选项 调度

重组方法

- ☐ 按递增的方式原地重组表(P)
- ☒ 重建表的影子副本(B)

表访问

- ☐ 不允许任何用户访问(N)
- ☒ 允许只读访问(R)
- ☐ 允许读写访问(W)

重组选项

- ☒ 使用现有的索引进行重组(I) WHU.DEPT_PK_ID
- ☒ 对临时数据使用系统临时空间(M) TEMPSPACE1
- ☐ 使用索引扫描(C)
- ☐ 重组长字段和大对象数据(L)
- ☒ 使用系统临时空间(G) TEMPSPACE1

确定 取消 显示命令(W) 帮助

10.4 组织数据库中的数据

➤ 生成统计信息**RUNSTATS**

- **RUNSTATS**收集的统计信息能用在两个方面，显示资料的物理构成和给**DB2**提供信息优化器，以便在执行**SQL**语句的时候选择最佳访问路径
- 这些特性包括记录的数量、页的数量、平均记录长度

RUNSTATS ON TABLE <table name>

10.4 组织数据库中的数据

➤ 重新联编REBIND

- **REBIND**程序支持利用系统编目表中的信息重新建立一个包，允许**SQL**应用程序通过优化器选择使用不同的访问路线
- 在运行了**REORG**或者**RUNSTATS**后运行**REBIND**程序

db2rbind sample -l db2rbind.out

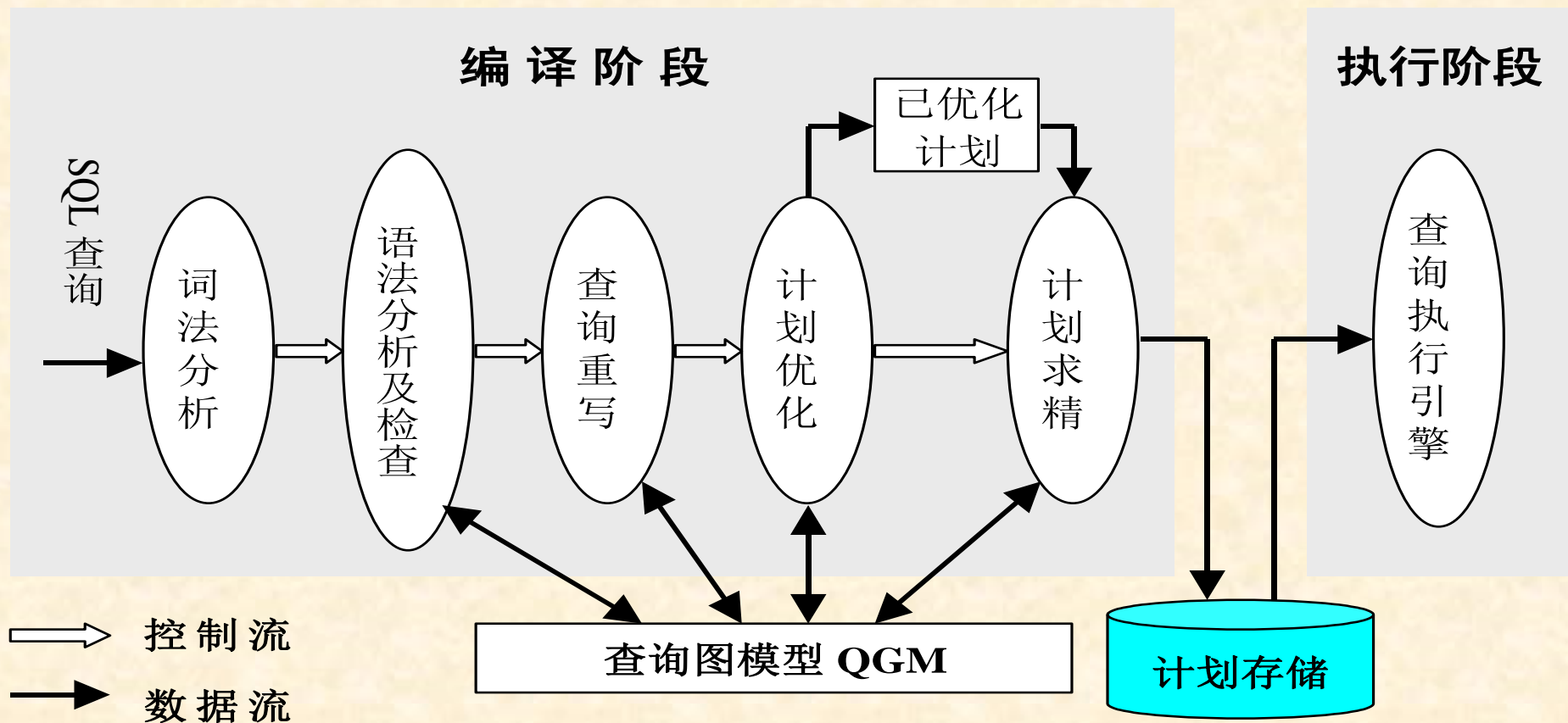
10.4 关系数据库的查询优化

- 从上个世纪70年代到80年代，在这近十年的研究过程中所获得的与查询优化有关的研究成果主要有：
 - 一个关系代数表达式可以有多种不同的有效等价表达式
 - 关系代数表达式的表示形式与其执行效率之间存在着必然的联系
 - 获取具有较高查询效率的表达式的算法

10.4 关系数据库的查询优化

➤ 查询优化的实现方法

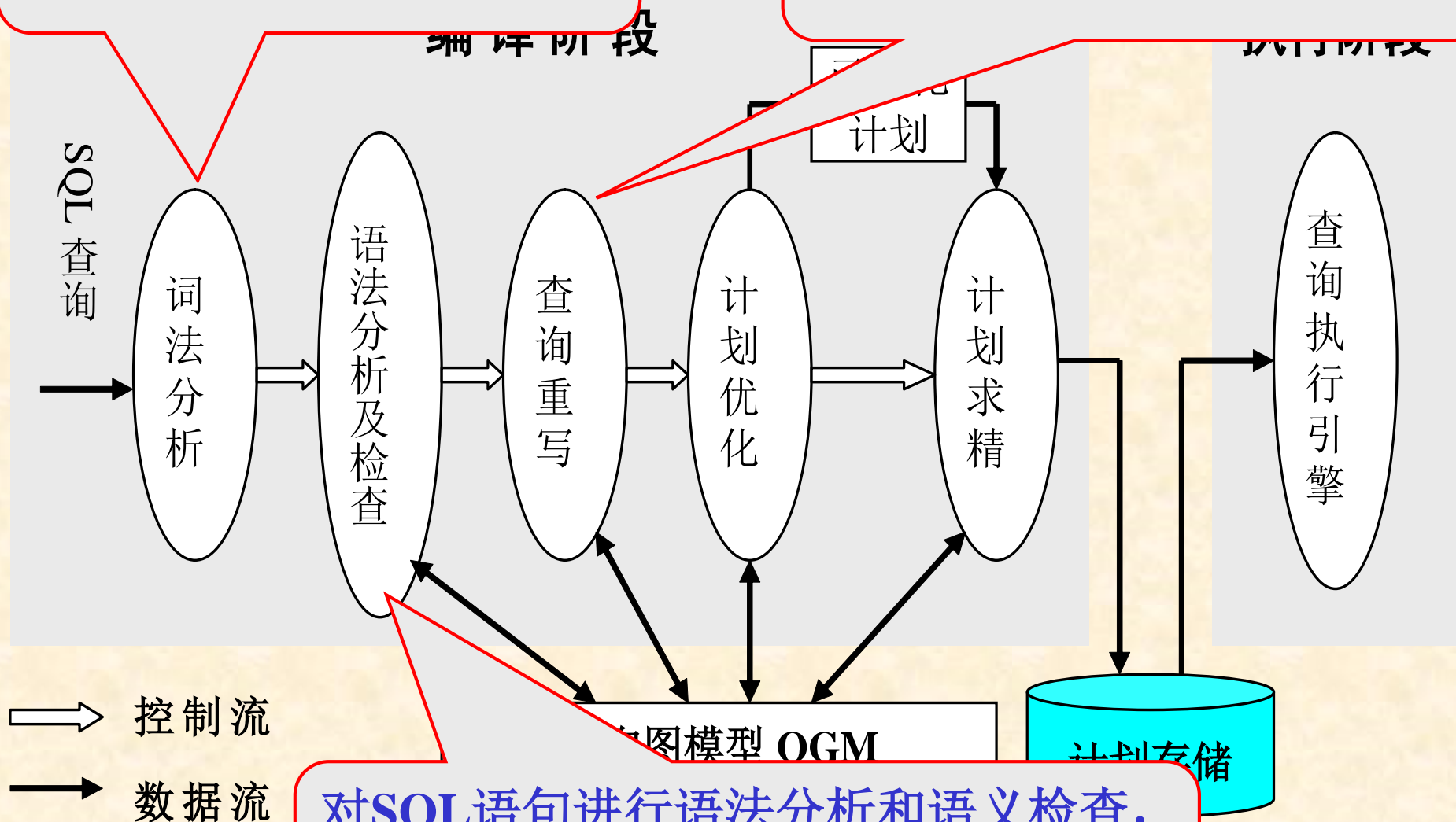
- 设置一个查询优化器，对用户输入的**SQL**查询语句，经优化器处理后产生优化后的查询表达式



查询优化处理流程

对SQL语句的保留字进行词法分析并转换成内部码

优化器将查询重写成为一种等价且效率较高的表示形式



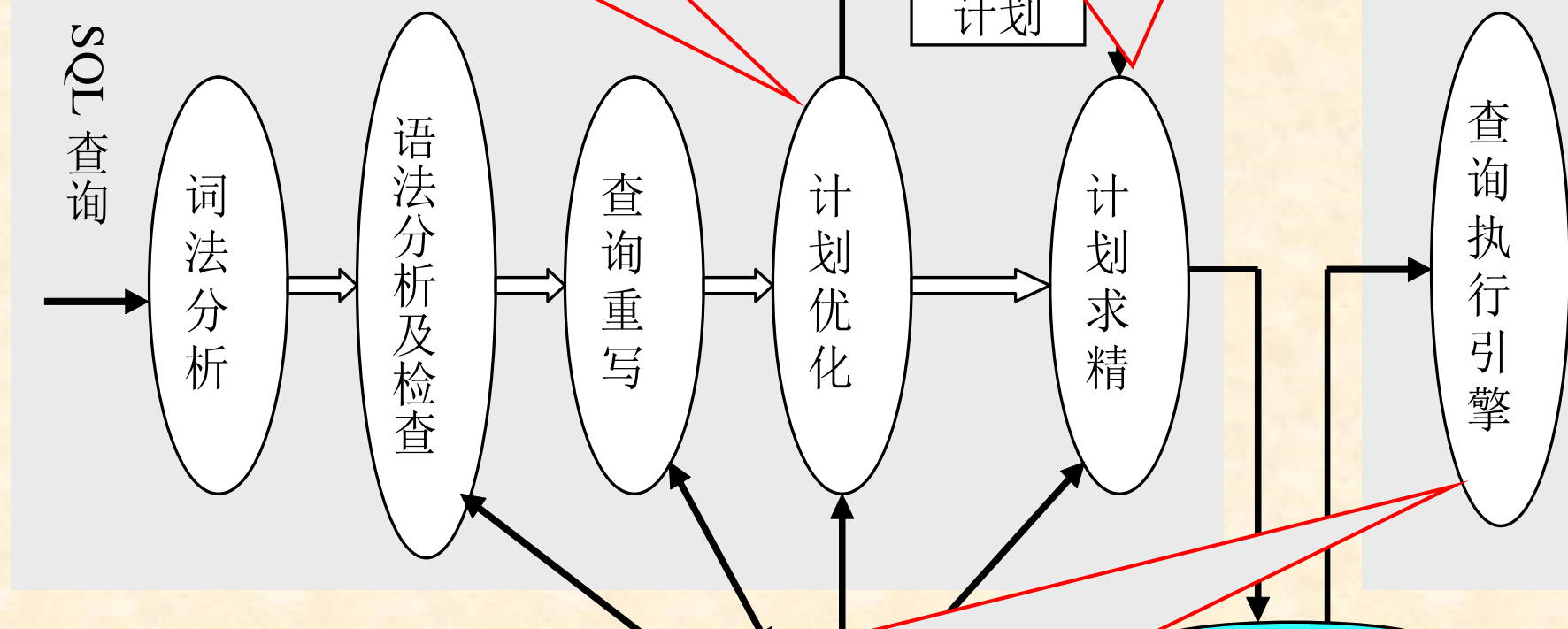
对SQL语句进行语法分析和语义检查，并将其转换成一种适合于内部使用的表示形式（查询图模型QGM）；

根据可用的存取路径生成
一组可能的执行计划

根据数据分布和统计信息评
估每一个执行计划的代价，
选取其中代价最小的一个

计划阶段

执行阶段



⇒ 控
→ 数

提供一组物理操作符（如：排序、顺序扫描、索引扫描、嵌套循环join、排序归并join等），用于执行具体的计划。计划在执行时还可做进一步的优化

诸

10.4 关系数据库的查询优化

例1 查找修读课程号为C₅的所有学生姓名

```
SELECT DISTINCT S.Sn  
FROM S, SC  
WHERE S.sno=SC.sno AND  
SC.cno='C5'
```

➤该查询语句可以有多种相互等价的关系代数表达式，如：

$$Q_1: \pi_{Sn}(\delta_{S.sno=SC.sno \wedge SC.cno='C_5'}(S \times SC))$$

$$Q_2: \pi_{Sn}(\delta_{SC.cno='C_5'}(S \bowtie SC))$$

$$Q_3: \pi_{Sn}(S \bowtie (\delta_{SC.cno='C_5'}(SC)))$$

10.4 关系数据库的查询优化

➤ 假设:

- 1) 设关系**S**中有**1000**个元组，关系**SC**中有**10000**个元组，其中修读**C₅**课程的元组数为**50**。
- 2) 每个物理磁盘块能存放**10**个**S**元组（关系**S**占用**100**个磁盘块），或**100**个**SC**元组（关系**SC**占用**100**个磁盘块）。
- 3) 内存有**6**个磁盘块大小的缓冲区，其中**5**块可存放**S**元组，**1**块存放**SC**元组。
- 4) 读/写一块磁盘的时间为**(1/20)s**，即每秒可读写**20**个磁盘块。
- 5) 为简化起见，所有内存操作所花时间忽略不计。

➤ 我们分别来估算一下上述的三个查询表达式在执行过程中的时间开销

$$Q_1: \pi_{S_n} (\delta_{S.sno=SC.sno \wedge SC.cno='C5'} (S \times SC))$$

Q_1 的执行步骤如下:

(1) $R1 = S \times SC$

- 执行两个关系的笛卡儿乘积。具体执行过程如下:
 - 一次读取5个磁盘块的学生关系中的元组到内存缓冲区 (约50个S元组)
 - 然后将学习关系SC中的数据依次读入内存缓冲区中, 并分别与内存缓冲区中的50个学生元组进行联接操作 (每次读写100个磁盘块, 共需读取20次)。
 - 共需读取磁盘块数: $100 + 100 \times 20 = 2100$
- 它们的笛卡儿乘积的结果元组有 10^7 个, 假设每个磁盘块可以存放约10条这样的元组, 则中间结果关系R1需占用 10^6 个磁盘块, 因而中间结果关系的保存需写磁盘块的数目是: 10^6
- 两部分总的~~时间开销~~: $(2100+10^6)/20 = 50105$ 秒

$$Q_1: \pi_{S_n} (\delta_{S.sno=SC.sno \wedge SC.cno='C5'} (S \times SC))$$

- （思考1）假设执行过程是：每次读取1个磁盘块的SC元组到内存缓冲，然后依次将关系S中的元组读入内存缓冲，并分别与内存中的SC元组进行联接操作。则这样的执行过程共需读取磁盘块的次数是多少？
- （思考2）假设：关系R和S分别有 D_R 和 D_S 个磁盘块，可使用的内存缓冲区分别是 M_R 和 M_S 块。执行关系R和关系S的笛卡儿乘积操作需要的磁盘读次数分别是：（不考虑笛卡儿乘积的结果元组的存储）

❖ 以关系R为外层循环，关系S为内层循环：

$$\bullet D_R + D_R \times D_S / M_R$$

❖ 以关系S为外层循环，关系R为内层循环：

$$\bullet D_S + D_R \times D_S / M_S$$

$$Q_1: \pi_{S_n} (\delta_{S.sno=SC.sno \wedge SC.cno='C5'} (S \times SC))$$

$$(2) \quad R2 = \delta_{S.sno=SC.sno \wedge SC.cno='C5'} (R1)$$

- 将步骤（1）得到的中间结果元组读入内存，并选择出符合条件的元组。
- 由于选择操作是在内存缓冲中执行的，且满足条件的元组只有**50**个，可以全部放在内存中。因此这一步执行所需要的时间开销仅仅是将中间关系**R1**中的元组读入内存，其时间开销是：

$$10^6 / 20 = 50000 \text{ 秒}$$

$$(3) \quad R3 = \pi_{S_n} (R2)$$

- 都是内存操作，其时间开销可以忽略不计

➤ Q_1 的全部查询时间为： $50105 + 50000 \approx 10^5$ 秒

$$Q_2: \pi_{S_n}(\delta_{SC.cno='C5'}(S \bowtie SC))$$

Q_2 的执行步骤如下:

(1) $R1 = S \bowtie SC$

- 执行两个关系的自然联接。其执行过程与 Q_1 类似，但中间结果元组只有 10^4 个。该步执行需要的时间开销是：

- 读取关系 S 和 SC 的磁盘块数：2100

- 保存中间结果的写磁盘块数： $10^4/10 = 1000$

❖ 两部分总的时间开销： $(2100+1000)/20 = 155$ 秒

(2) $R2 = \delta_{SC.cno='C5'}(R1)$

- 执行时间为读取中间关系 $R1$ 的时间，共需：

$1000/20 = 50$ 秒

➤ **(3) $R3 = \pi_{S_n}(R2)$ 时间忽略不计**

➤ **Q_2 的全部查询时间为： $155 + 50 = 205$ 秒**

$$Q_3: \pi_{S_n} (S \bowtie (\delta_{SC.cno='C5'} (SC)))$$

Q_3 的执行步骤如下:

(1) $R1 = \delta_{SC.cno='C5'} (SC)$

- 单表上的选择操作，只需读**SC**表一遍（**100**个磁盘块）。其结果元组只有**50**个，可以全部放在内存中，不需要写中间结果的时间。因此总的开销为： **$100/20 = 5$ 秒**

(2) $R2 = S \bowtie R1$

- 关系**R1**的元组已经全部在内存，因此只需读一遍**S**表（**100**个磁盘块），所花的时间是： **$100/20 = 5$ 秒**

➤ (3) $R3 = \pi_{S_n} (R2)$ 时间忽略不计

➤ Q_3 的全部查询时间为： **$5 + 5 = 10$ 秒**

10.4 查询优化的可能性

➤ 三个查询的时间开销比较

	总的执行时间	中间结果需要的 磁盘块数
Q_1	10^5 秒	10^6
Q_2	205 秒	10^3
Q_3	10 秒	0

➤ **结论：**合理选取查询命令的关系代数表达式，可以获得较高的查询效率

10.4 关系代数的等价变换规则

- 查询优化的关键是选择合理的等价表达式。为此，需要一套完整的关系代数表达式的等价变换规则，下面我们给出一组常用的等价变换规则

10.4 关系代数的等价变换规则

1. 交换律

$$E_1 \times E_2 \equiv E_2 \times E_1 \quad (6.1)$$

$$E_1 \bowtie E_2 \equiv E_2 \bowtie E_1 \quad (6.2)$$

$$E_1 \bowtie_F E_2 \equiv E_2 \bowtie_F E_1 \quad (6.3)$$

2. 结合律

$$(E_1 \times E_2) \times E_3 \equiv E_1 \times (E_2 \times E_3) \quad (6.4)$$

$$(E_1 \bowtie E_2) \bowtie E_3 \equiv E_1 \bowtie (E_2 \bowtie E_3) \quad (6.5)$$

$$(E_1 \bowtie_{F1} E_2) \bowtie_{F2} E_3 \equiv E_1 \bowtie_{F1} (E_2 \bowtie_{F2} E_3) \quad (6.6)$$

10.4 关系代数的等价变换规则

3. 投影串接定律

- 设 E 为关系代数表达式, A_i ($i=1,2,\dots,n$), B_j ($j=1,2,\dots,m$)为属性变元, $\{A_1, A_2, \dots, A_n\} \subset \{B_1, B_2, \dots, B_m\}$, 此时必有:

$$\pi_{A_1, A_2, \dots, A_n}(\pi_{B_1, B_2, \dots, B_m}(E)) \equiv \pi_{A_1, A_2, \dots, A_n}(E) \quad (6.7)$$

4. 选择的串接定律

$$\delta_{F_1}(\delta_{F_2}(E)) \equiv \delta_{F_1 \wedge F_2}(E) \quad (6.8)$$

10.4 关系代数的等价变换规则

5. 选择和投影的交换律

- 如果选择条件 F 所使用的属性全在 A_1, A_2, \dots, A_n 中，则有：

$$\delta_F(\pi_{A_1, A_2, \dots, A_n}(E)) \equiv \pi_{A_1, A_2, \dots, A_n}(\delta_F(E)) \quad (6.9)$$

- 如果选择条件 F 所使用的属性全在 $A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m$ 中，则有：

$$\pi_{A_1, A_2, \dots, A_n}(\delta_F(E)) \equiv \pi_{A_1, A_2, \dots, A_n}(\delta_F(\pi_{A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m}(E))) \quad (6.10)$$

10.4 关系代数的等价变换规则

6. 选择与笛卡尔乘积的交换律

- 如果F中的属性都是E₁的属性，则有：

$$\delta_F(E_1 \times E_2) \equiv \delta_F(E_1) \times E_2 \quad (6.11)$$

- 如果F=F₁∧F₂,且F₁的属性全部属于E₁,F₂的属性全部属于E₂,则有：

$$\delta_F(E_1 \times E_2) \equiv \delta_{F_1}(E_1) \times \delta_{F_2}(E_2) \quad (6.12)$$

- 如果F=F₁∧F₂∧F₃,且F₁属性只属于E₁,F₂属性只属于E₂,F₃属性属于E₁和E₂,则有：

$$\delta_F(E_1 \times E_2) \equiv \delta_{F_3}(\delta_{F_1}(E_1) \times \delta_{F_2}(E_2)) \quad (6.13)$$

- 如果将‘笛卡尔乘积’换成‘自然联接’，则上述规则是否还成立？

10.4 关系代数的等价变换规则

➤ 选择与自然联接的交换律

- 如果F中的属性都是E₁的属性，则有：

$$\delta_F(E_1 \bowtie E_2) \equiv \delta_F(E_1) \bowtie E_2 \quad (6.11.1)$$

- 如果F中的属性既是E₁的属性，也是E₂的属性，则有：

$$\delta_F(E_1 \bowtie E_2) \equiv \delta_F(E_1) \bowtie \delta_F(E_2) \quad (6.11.2)$$

- 如果F=F₁∧F₂，且F₁的属性全部属于E₁，F₂的属性全部属于E₂，则有：

$$\delta_F(E_1 \bowtie E_2) \equiv \delta_{F_1}(E_1) \bowtie \delta_{F_2}(E_2) \quad (6.12.1)$$

- 如果F=F₁∧F₂∧F₃，且F₁属性只属于E₁，F₂属性只属于E₂，F₃属性属于E₁和E₂，则有：

$$\delta_F(E_1 \bowtie E_2) \equiv \delta_{F_3}(\delta_{F_1}(E_1) \bowtie \delta_{F_2}(E_2)) \quad (6.13.1)$$

10.4 关系代数的等价变换规则

7. 选择与并运算的交换律

- 设 E_1 及 E_2 为关系代数表达式,且具有相同的结果关系模式,则有:

$$\delta_F(E_1 \cup E_2) \equiv \delta_F(E_1) \cup \delta_F(E_2) \quad (6.14)$$

8. 选择与差运算的交换律

- 设 E_1 及 E_2 为关系代数表达式,且具有相同的结果关系模式,则有:

$$\delta_F(E_1 - E_2) \equiv \delta_F(E_1) - \delta_F(E_2) \quad (6.15)$$

10.4 关系代数的等价变换规则

9. 投影与笛卡尔乘积交换律

- 设 E_1 及 E_2 为关系代数表达式, A_1, A_2, \dots, A_n 是 E_1 的属性, B_1, B_2, \dots, B_m 是 E_2 的属性,则有:

$$\pi_{A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m}(E_1 \times E_2) \equiv \pi_{A_1, A_2, \dots, A_n}(E_1) \times \pi_{B_1, B_2, \dots, B_m}(E_2) \quad (6.16)$$

- 如果将‘笛卡尔乘积’换成‘自然联接’，则该规则是否仍然成立？

10.4 关系代数的等价变换规则

10. 投影与并运算的交换律

- 设 E_1 及 E_2 为关系代数表达式,且具有相同的结果关系模式,则有:

$$\pi_{A_1, A_2, \dots, A_n}(E_1 \cup E_2) \equiv \pi_{A_1, A_2, \dots, A_n}(E_1) \cup \pi_{A_1, A_2, \dots, A_n}(E_2) \quad (6.17)$$

- 如果将‘并’运算换成‘差’运算,则该规则是否仍然成立?

10.4 关系代数的等价变换规则

11. 笛卡尔乘积与连接运算的转换律

- 设 E_1, E_2 为关系代数表达式, E_1 的属性变元为 A_1, A_2, \dots, A_n , E_2 的属性变元为 B_1, B_2, \dots, B_m , F 为形如 $E_1.A_i \theta E_2.B_j$ 所组成的合取式, 则有:

$$\delta_F(E_1 \times E_2) = E_1 \bowtie_F E_2 \quad (6.18)$$

10.4 关系代数的等价变换规则

例2 我们可以将例1中表达式 Q_1 转换成 Q_2 ，同时也可以将 Q_2 转换成 Q_3 。

- 从 Q_1 到 Q_2

$$\begin{aligned} & \pi_{S_n}(\delta_{S.sno=SC.sno \wedge SC.cno='C5'}(S \times SC)) \\ \equiv & \pi_{S_n}(\delta_{SC.cno='C5'}(\delta_{S.sno=SC.sno}(S \times SC))) \quad (\text{规则6.8}) \\ \equiv & \pi_{S_n}(\delta_{SC.cno='C5'}(S \bowtie SC)) \quad (\text{自然联接的定义}) \end{aligned}$$

- 从 Q_2 到 Q_3

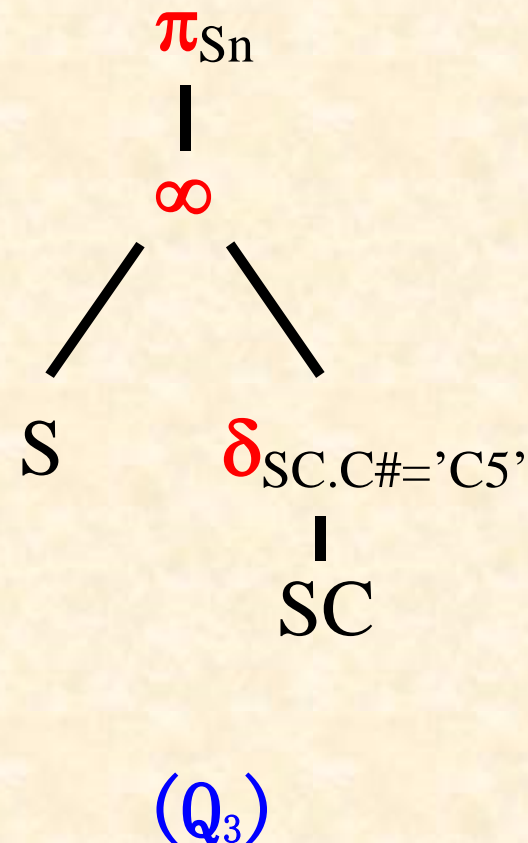
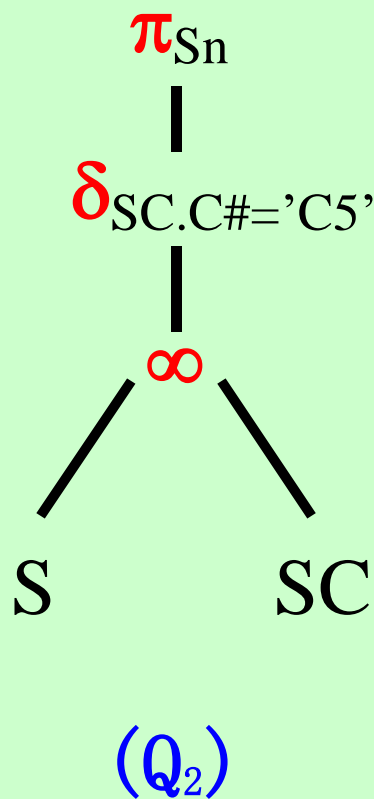
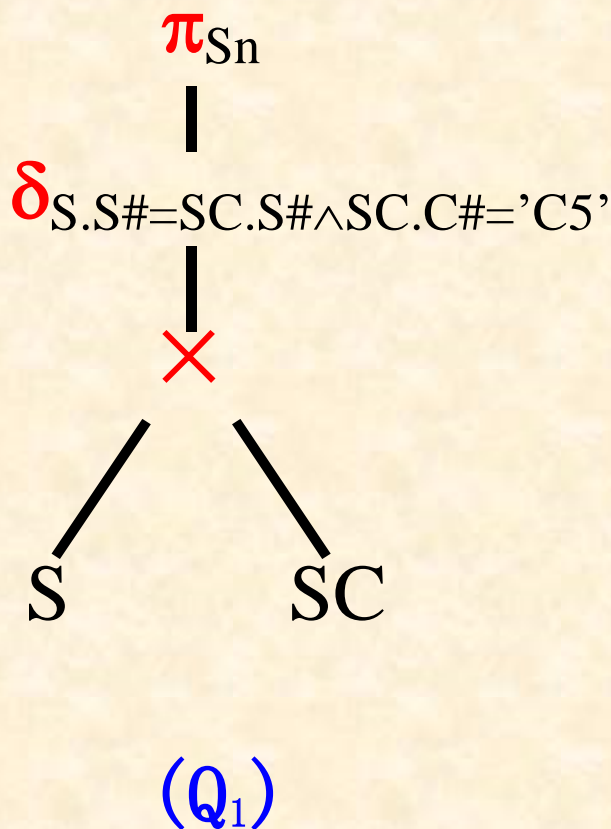
$$\begin{aligned} & \pi_{S_n}(\delta_{SC.cno='C5'}(S \bowtie SC)) \\ \equiv & \pi_{S_n}(S \bowtie \delta_{SC.cno='C5'}(SC)) \quad (\text{规则6.11}) \end{aligned}$$

10.4 查询优化策略与算法

一、优化策略

- 在执行查询优化之前，我们首先将关系代数表达式转换成一棵语法树，该树的叶结点是需要操作的基表（关系），树中的结点则是基表上的操作
- 然后再根据一定的优化策略对该语法树进行优化

10.4 查询优化策略与算法

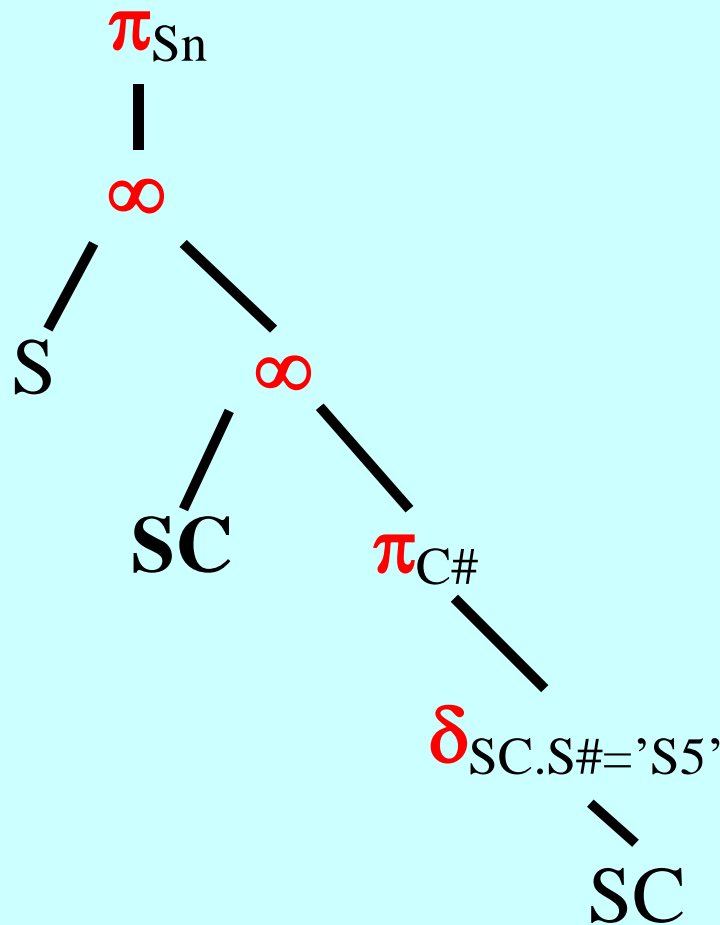


➤例1：三个关系代数表达式的语法树

10.4 查询优化策略与算法

➤ 例：至少修了S5所修读的一门课程的学生们的姓名。

$Q_4: \pi_{Sn} (S \bowtie (SC \bowtie \pi_{cno} (\delta_{SC.sno='S5'} (SC))))$



10.4 查询优化策略与算法

1. 下推选择

- 选择运算应尽可能先做。
- 如：规则6.9、6.11~13、6.14、6.15，可以用规则右边的表达式来替换左边的表达式。
- 例：检索计算机(CS)系选修数据库(DB)课程学生的学号和成绩。

$$\pi_{sno,G} (\delta_{S.Sd='CS' \wedge C.Cn='DB'} ((S \bowtie SC) \bowtie C))$$

❖ 利用规则6.12.1将其转换为：

$$\pi_{sno,G} (\delta_{S.Sd='CS'} (S \bowtie SC) \bowtie \delta_{C.Cn='DB'} (C))$$

❖ 利用规则6.11.1将其转换为：

$$Q_5: \pi_{sno,G} ((\delta_{S.Sd='CS'} (S) \bowtie SC) \bowtie \delta_{C.Cn='DB'} (C))$$

10.4 查询优化策略与算法

- 在某些情况下（如使用视图的查询），可能需要先上移某个选择操作，然后再将其下推到所有可能的分支

- 例：设有两个关系：**Course(Cno,Cname,Year,Hours)**和**Teacher(Tno,Tname,Age,Cno,Year)**,在关系**Course**上建立一个由在**2002**年所开设的课程构成的视图：

CREATE VIEW C2002 AS

» **select * from Course where Year = 2002**

- 在该视图上执行如下查询(Q_6): 在**2002**年担任了教学任务的教师的姓名及其课程名。其优化过程是：

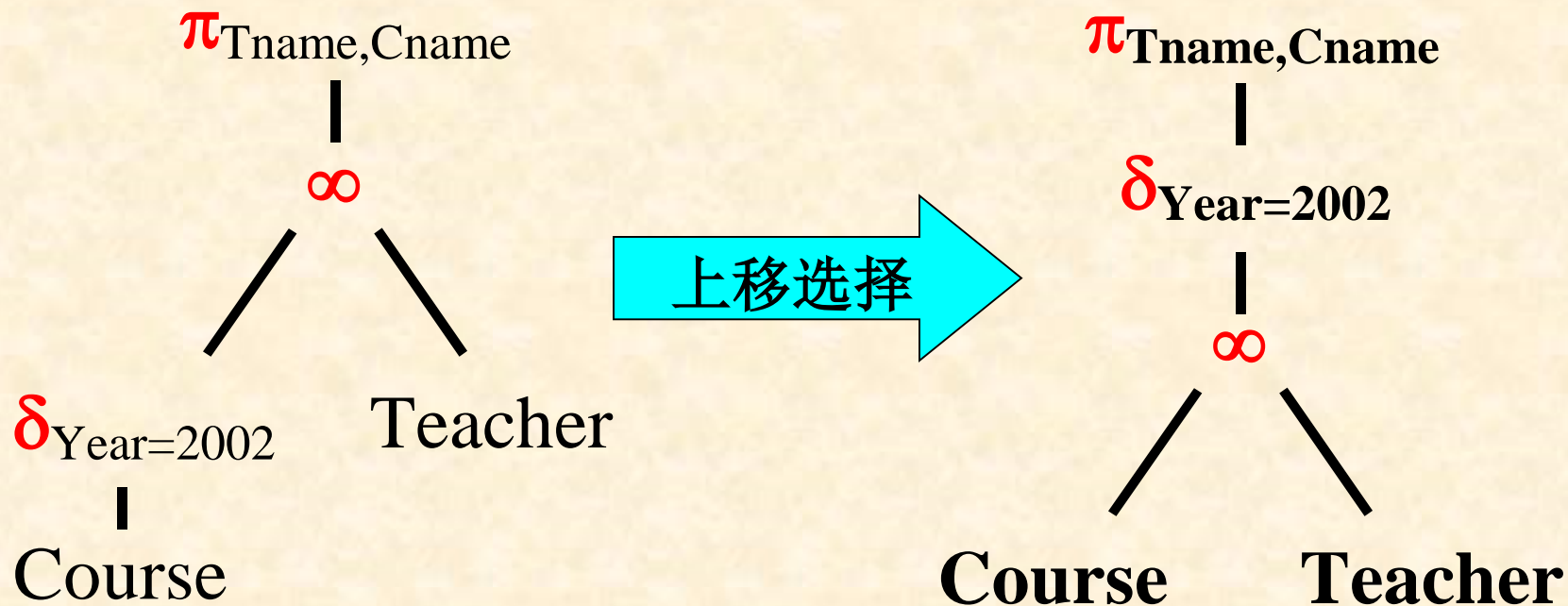
$\pi_{Tname,Cname}(C2002 \bowtie Teacher)$

$\pi_{Tname,Cname}(\delta_{Year=2002}(Course) \bowtie Teacher)$

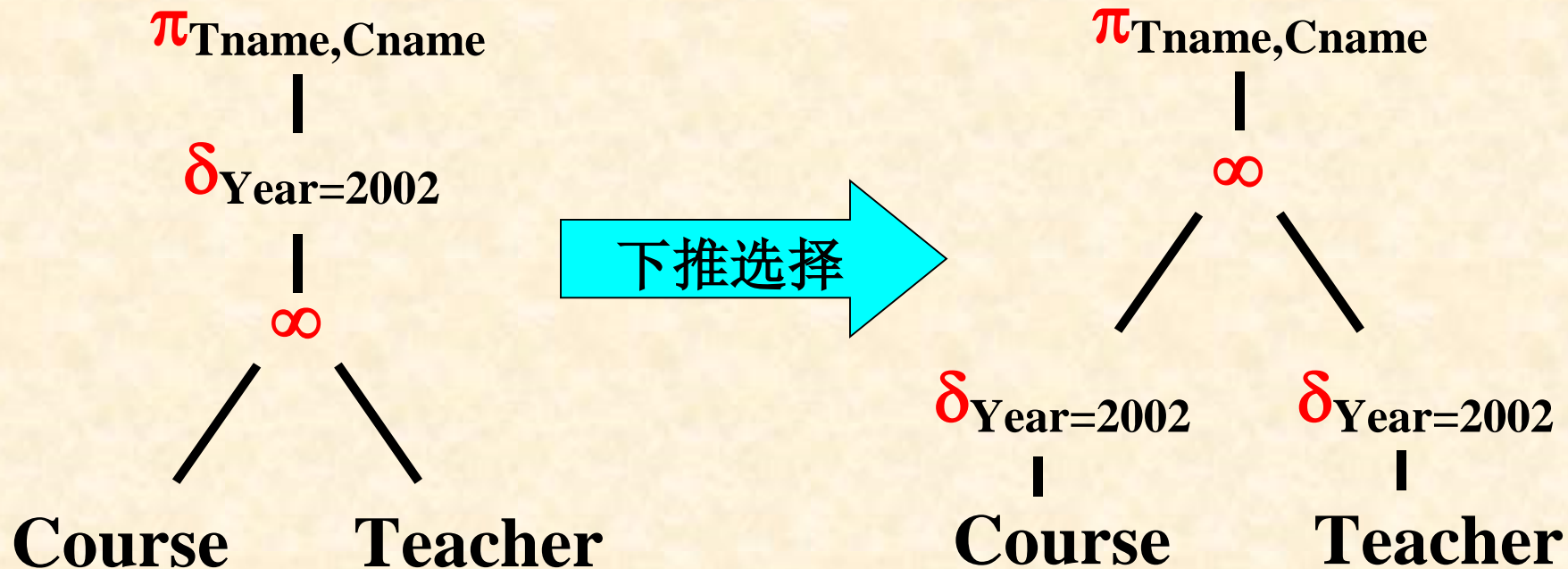
$\pi_{Tname,Cname}(\delta_{Year=2002}(Course \bowtie Teacher))$

$\pi_{Tname,Cname}(\delta_{Year=2002}(Course) \bowtie \delta_{Year=2002}(Teacher))$

10.4 查询优化策略与算法



10.4 查询优化策略与算法



10.4 查询优化策略与算法

2. 与投影操作有关的策略

- 1) 同一对象的投影与选择运算应尽可能同时做（合并相邻的投影与选择运算的执行）；
- 2) 将投影与其前后的二元运算结合起来做（合并投影和相邻的二元运算）；
 - ❖ 上述的合并是执行时间上的合并，在语法树上并没有反映出来。
 - ❖ 例如：Q₄
- 3) 下推投影
 - 不同于下推选择。下推投影是在一个已经存在的投影运算之下插入一个新的投影运算（通过新插入的投影运算所消除的属性是其后的操作不会用到的）

10.4 查询优化策略与算法

➤ 相关规则有:

$$\blacksquare \pi_L(R \bowtie S) \equiv \pi_L(\pi_M(R) \bowtie \pi_N(S)) \quad (6.19)$$

❖ 其中: **M**是由出现在关系**R**且出现在**L**中的属性加上关系**R**和**S**的公共属性所构成的(而**N**也类似)

$$\blacksquare \pi_L(R \bowtie_F S) \equiv \pi_L(\pi_M(R) \bowtie_F \pi_N(S)) \quad (6.20)$$

$$\blacksquare \pi_L(R \times S) \equiv \pi_L(\pi_M(R) \times \pi_N(S)) \quad (6.21)$$

$$\blacksquare \pi_L(R \cup S) \equiv \pi_L(R) \cup \pi_L(S) \quad (6.22)$$

▪ 但是:

$$\pi_L(R \cap S) \equiv \pi_L(R) \cap \pi_L(S)$$

$$\pi_L(R - S) \equiv \pi_L(R) - \pi_L(S)$$

10.4 查询优化策略与算法

➤ [例Q4] 一个比较烦琐的写法是：

$$\pi_{S_n}(\pi_{sno, S_n}(S) \bowtie \pi_{sno}(\pi_{sno, cno}(SC) \bowtie \pi_{cno}(\delta_{SC.sno='S5'}(SC))))$$

❖ 考虑到策略2-2的要求，可以消去其中的两个投影运算： π_{sno, S_n} 和 $\pi_{sno, cno}$ ，但另三个投影运算是不能省略的。

■ 综上所述：

- ❖ 如果用于投影的对象是一个基表（有存储数据）时，新增的投影操作没有多少意义（有时反而会适得其反，如无法使用建立在基表中的索引）；
- ❖ 如果是其它中间运算结果（或视图），则新引入的投影操作会降低中间结果的数据量，因而是有意义的

10.4 查询优化策略与算法

3. 涉及联接和自然联接的策略

- 将选择运算与其下面的笛卡儿乘积组合成联接运算。
 - ❖ 规则 6.18
 - ❖ 规则 6.23: $R \bowtie S \equiv \pi_L(\delta_F(R \times S))$
- 例：从 Q_1 到 Q_2
 - ❖ 规则6.7：插入一些新的投影运算
 - ❖ 规则6.9：交换投影运算与选择运算的顺序
 - ❖ 规则6.8：拆分选择运算
 - ❖ 规则6.9：交换投影运算与选择运算的顺序
 - ❖ 规则6.23：合并成自然联接运算

10.4 查询优化策略与算法

4. 消除重复结果元组的策略

- 定义一个新操作符 $\sigma(R)$: 消除 R 中的重复元组
- 相关规则:

❖ 如果 R 是一个含有主码定义的关系, 或者是一个经过统计计算得到的结果, 则: $\sigma(R) \equiv R$ (6.24)

❖ $\sigma(R \times S) \equiv \sigma(R) \times \sigma(S)$ (6.25)

❖ $\sigma(R \bowtie S) \equiv \sigma(R) \bowtie \sigma(S)$ (6.26)

❖ $\sigma(R \bowtie_F S) \equiv \sigma(R) \bowtie_F \sigma(S)$ (6.27)

❖ $\sigma(\delta_F(R)) \equiv \delta_F(\sigma(R))$ (6.28)

❖ 但是: 去重运算(σ)与投影运算(π)不能互换

❖ 在这里没有定义 \cap 、 \cup 、 $-$ 与 σ 运算符的优化策略。

❖ 总的优化策略: 尽量下推 σ 运算

10.4 查询优化策略与算法

5. 涉及分组统计的策略

- 定义另一个新操作符 $\gamma_L(R)$: L 是分组统计结果所需的属性
- 相关规则:

$$\diamond \sigma(\gamma_L(R)) \equiv \gamma_L(R) \quad (6.29)$$

$$\diamond \gamma_L(R) \equiv \gamma_L(\pi_M(R)) \quad (6.30)$$

- 如果统计操作不受 R 中重复元组的影响, 则:

$$\gamma_L(R) \equiv \gamma_L(\sigma(R)) \quad (6.31)$$

- 优化策略: 在允许的情况下, 引入新的投影和选择运算

10.4 查询优化策略与算法

6. 找出公共子表达式

- 以避免重复计算。

7. 执行前的预处理

- 文件排序
- 建立针对联接属性的索引

10.4 查询优化策略与算法

➤ 优化算法

- 1) 将关系代数表达式转化成一棵语法树;
- 2) 用规则6.8将形如 $\delta_{F_1 \wedge F_2 \wedge \dots \wedge F_n} (E)$ 的表达式变换为:
$$\delta_{F_1} (\delta_{F_2} (\dots (\delta_{F_n} (E))))$$
- 3) 用规则6.8 ~ 规则6.15将选择运算尽量向内深入靠近关系(即向下移动, 靠近语法树的叶结点);
- 4) 用规则6.7、6.9、6.10、6.16、6.17将投影运算尽量向内深入靠近关系, 其具体方法是:
 - ❖ 用规则 6.7合并一些投影运算;
 - ❖ 用规则 6.9、6.16、6.17 将投影运算向内深入靠近关系;
 - ❖ 用规则 6.10 插入一些新的投影运算。
- 5) 用规则6.7 ~ 规则6.10将一组投影、选择运算串接成单个投影、选择或一个选择后紧跟一个投影运算;

10.4 查询优化策略与算法

➤ 优化算法（续）

- 5) 用规则6.18将笛卡尔乘积与选择运算结合成联接运算;
- 6) 将语法树按照以一个二元运算为核心的原则进行分组:
 - ❖ 由一个叶结点（关系）及其所有是一元运算的直接祖先结点构成一个新的叶结点;
 - ❖ 以一个二元运算结点 N 为中心，由结点 N 及其两个子女结点构成一个结点集合 S （其子女结点有可能是另一棵子树）;
 - ❖ 从结点 N 向上按序寻找其祖先结点 P ，如果 P 是一元运算，则将结点 P 加入到结点集合 S 中，直至碰到第一个是二元运算的祖先结点为止;
 - ❖ 由结点集合 S 中的结点可以构成该语法树的一棵子树。
- 7) 为上述的每一棵子树生成一个执行程序，以完成该子树所定义的操作。所有的子树构成一个层次结构，同一层子树的执行顺序是任意的