第三篇 系统篇

- 讨论数据库管理系统中查询处理和事务管理的基本概念和基础知识
  - 关系查询处理和查询优化
  - 数据库恢复
  - 并发控制

数据库系统概论
An Introduction to Database System

第九章 关系系统及其查询优化

中国人民大学信息学院 陈红

第九章　关系系统及其查询优化

关系系统及其查询优化（续）

- 本章目的：
  - RDBMS 的查询处理步骤
  - 查询优化的概念
  - 基本方法和技术

- 查询优化分类：
  - 代数优化：指关系代数表达式的优化
  - 物理优化：指存取路径和底层操作算法的选择

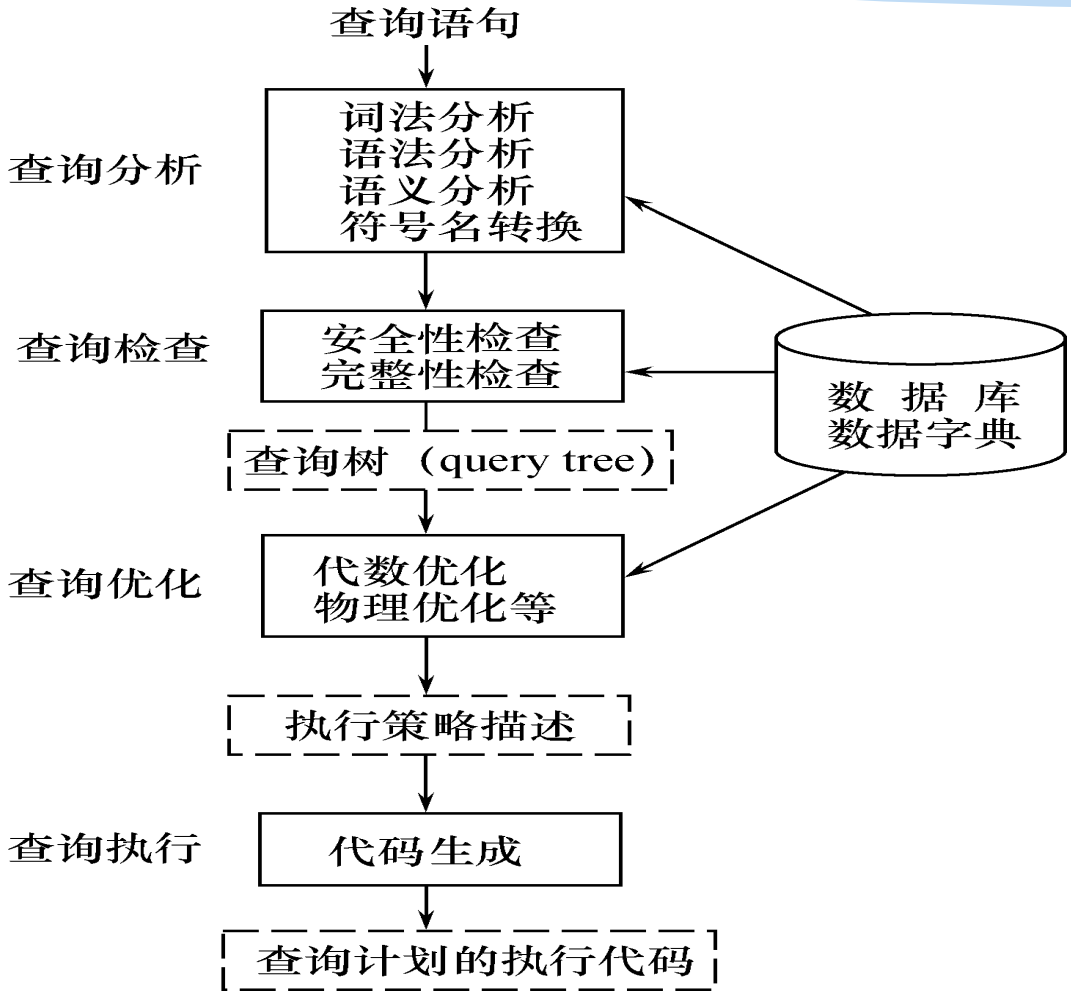
9.1 关系数据库系统的查询处理

- 9.1.1 查询处理步骤
- 9.1.2 实现查询操作的算法示例

9.1.1 查询处理步骤

- RDBMS 查询处理阶段：
  - 1. 查询分析
  - 2. 查询检查
  - 3. 查询优化
  - 4. 查询执行

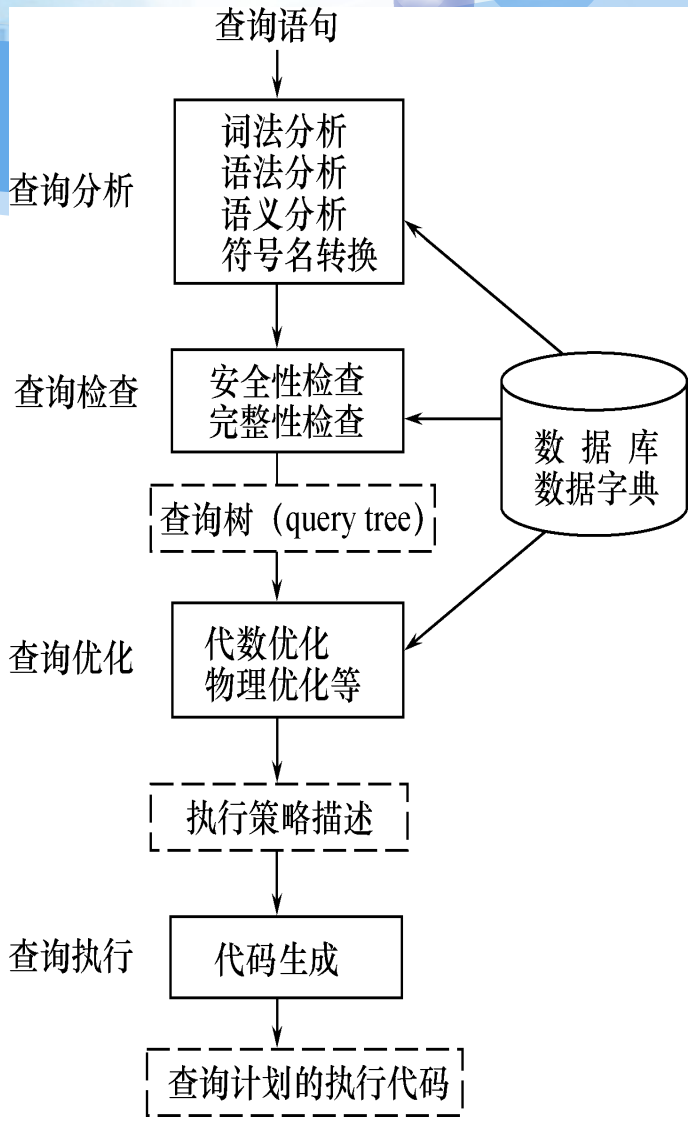
查询处理步骤（续）



查询处理步骤

1. 查询分析

- 查询分析的任务：对查询语句进行扫描，进行词法分析、语法分析和语义检查
- 词法分析：从查询语句中识别出正确的语言符号
- 语法分析：进行语法检查，生成语法分析树

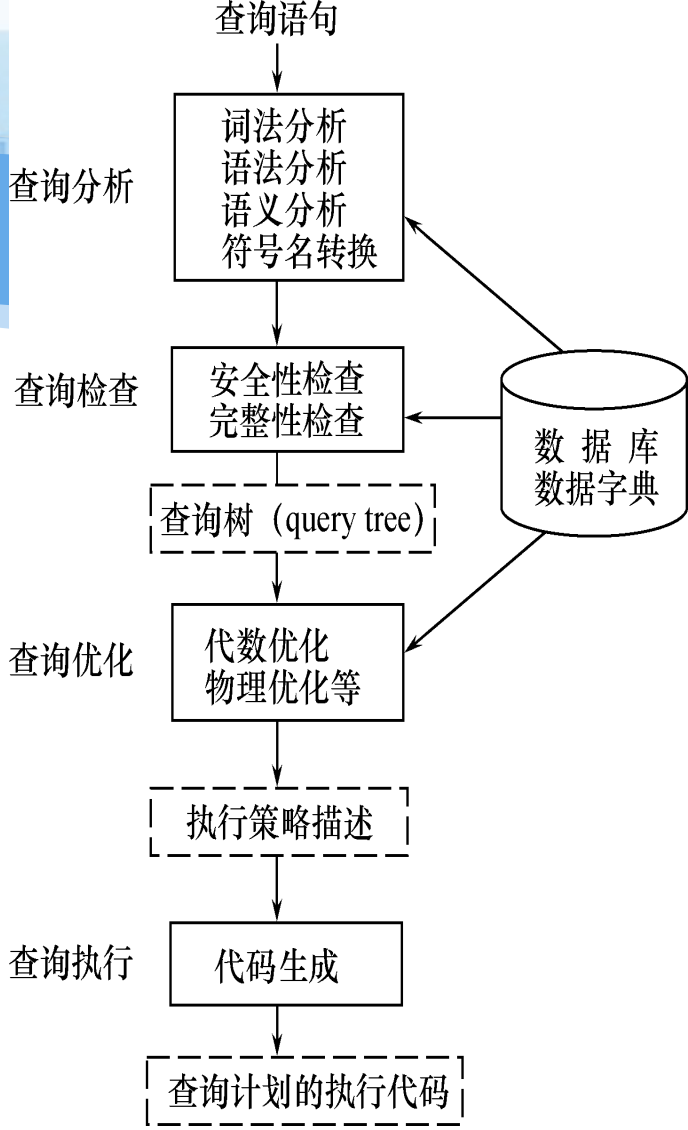


查询分析

- 语义检查的主要内容
  - 检查关系的使用
    - FROM 子句中出现的关系必须是该查询所对应模式中的关系或视图。
  - 检查与解析属性的使用
    - 在 SELECT 子句或 WHERE 子句中出现的各个属性必须是 FROM 子句中某个关系或视图的属性。
    - 当 FROM 子句中有多个关系 (视图) 时，往往通过给属性加上它所引用的关系名来解析每个属性，同时检查二义性。
  - 检查类型
    - 所有属性的类型必须与其使用相适应。
  - 在此过程中把数据库对象的外部名称转换为内部表示

2. 查询检查

- 查询检查的任务
  - 合法权检查
  - 完整性检查
  - 视图转换
- 根据数据字典中的用户权限定义对用户的存取权限进行检查
- 根据数据字典中的完整性约束定义进行静态完整性约束检查
  - 与值相关的约束检查在查询执

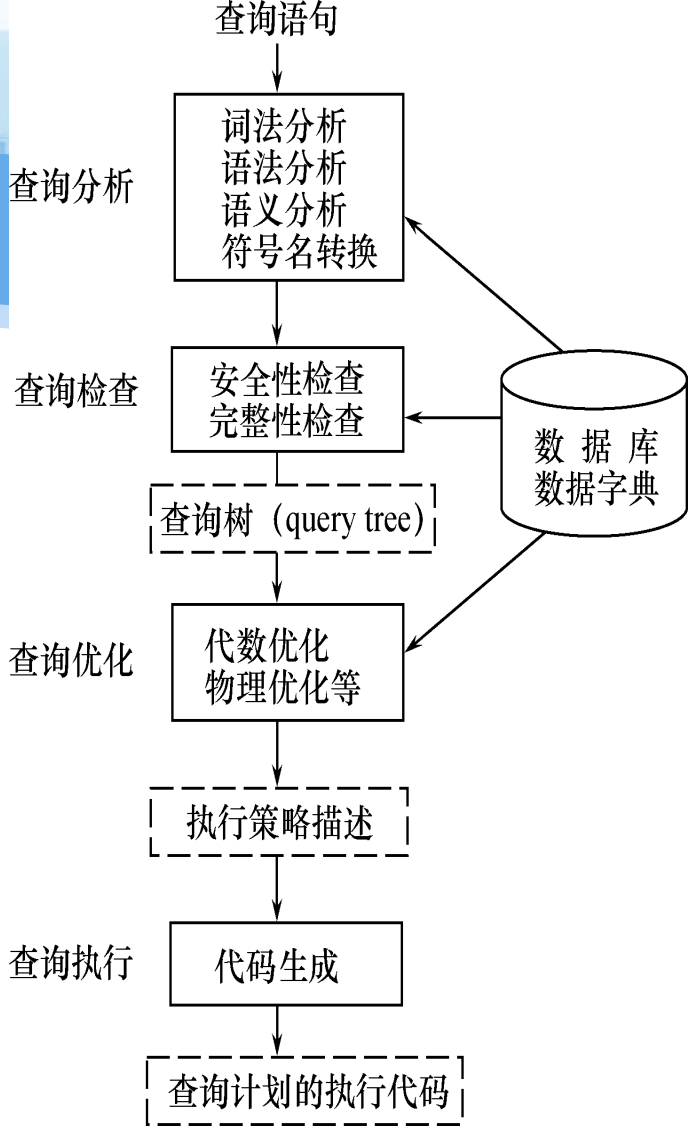


查询检查

- 视图转换方法
  - 视图消解
    - 在 from 列表中有用到视图的地方，用描述该视图的语法树来替换。该语法树由视图的定义得到。
  - 实体化视图
    - from 列表中引用视图时，生成新的查询，该查询将视图实体化为临时表。
    - 在 from 列表中用到视图的地方，用该临时表替换。

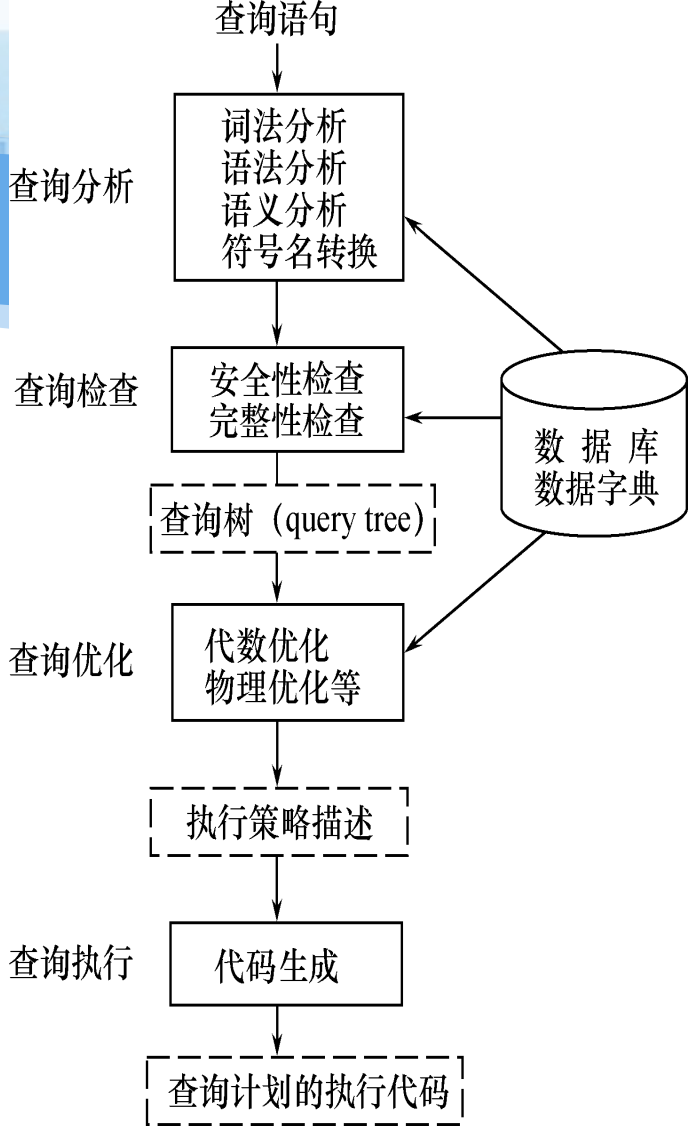
3. 查询优化

- 查询优化：选择一个高效执行的查询处理策略
- 查询优化分类：
  - 代数优化 / 逻辑优化：指关系代数表达式的优化
  - 物理优化：指存取路径和底层操作算法的选择
- 查询计划的选择依据：
  - 基于规则 (rule based)
  - 基于代价 (cost based)
  - 基于语义 (semantic based)



4. 查询执行

- 依据优化器得到的执行策略生成查询计划
- 代码生成器 (code generator) 生成执行查询计划的代码
- 两种执行方法
  - 自顶向下
  - 自底向上



9.1 关系数据库系统的查询处理

- 9.1.1 查询处理步骤
- 9.1.2 实现查询操作的算法示例

9.1.2 实现查询操作的算法示例

- 一、 选择操作的实现
- 二、 连接操作的实现

一、选择操作的实现

- 选择操作典型实现方法：
  - 1. 全表扫描方法 (Table Scan)
    - 对查询的基本表顺序扫描，逐一检查每个元组是否满足选择条件，把满足条件的元组作为结果输出
    - 适合小表，不适合大表
  - 2. 索引 (或散列) 扫描方法 (Index Scan)
    - 适合于选择条件中的属性上有索引 (例如 B+ 树索引或 Hash 索引)
    - 通过索引先找到满足条件的元组主码或元组指针，再通过元组指针直接在查询的基本表中找到元组

选择操作的实现（续）

例： Select * from student ;

算法：全表扫描

选择操作的实现（续）

例：

Select * from student where Sno = '200215121'

假设 Sno 上有索引（或 Sno 是散列码）

算法：

- 使用索引（或散列）得到 Sno 为 ‘200215121’ 元组的指针
- 通过元组指针在 student 表中检索到该学生

选择操作的实现（续）

例：

Select * from student where Sage>20 ;

假设 Sage 上有 B+ 树索引

算法：

- 使用 B+ 树索引找到 Sage = 20 的索引项，以此为入口点在 B+ 树的顺序集上得到 Sage>20 的所有元组指针
- 通过这些元组指针到 student 表中检索到所有年龄大于 20 的学生。

选择操作的实现（续）

例： Select * from student where Sdept = 'CS' AND Sage>20；
假设 Sdept 和 Sage 上都有索引

- 算法一：分别用 Index Scan 找到 Sdept = ’ CS’ 的一组元组指针和 Sage>20 的另一组元组指针
  - 求这 2 组指针的交集
  - 到 student 表中检索
  - 得到计算机系年龄大于 20 的学生

- 算法二：找到 Sdept = ‘ CS’ 的一组元组指针，
  - 通过这些元组指针到 student 表中检索
  - 并对得到的元组检查另一些选择条件 (如 Sage>20) 是否满足
  - 把满足条件的元组作为结果输出。

二、 连接操作的实现

- 连接操作是查询处理中最耗时的操作之一
- 本节只讨论等值连接(或自然连接)最常用的实现算法

- ［例 2］ SELECT * FROM Student， SC

  WHERE Student.Sno=SC.Sno；

连接操作的实现（续）

- 1. 嵌套循环方法 (nested loop)
- 2. 排序 - 合并方法 (sort-merge join 或 merge join)
- 3. 索引连接 (index join) 方法
- 4. Hash Join 方法

连接操作的实现（续）

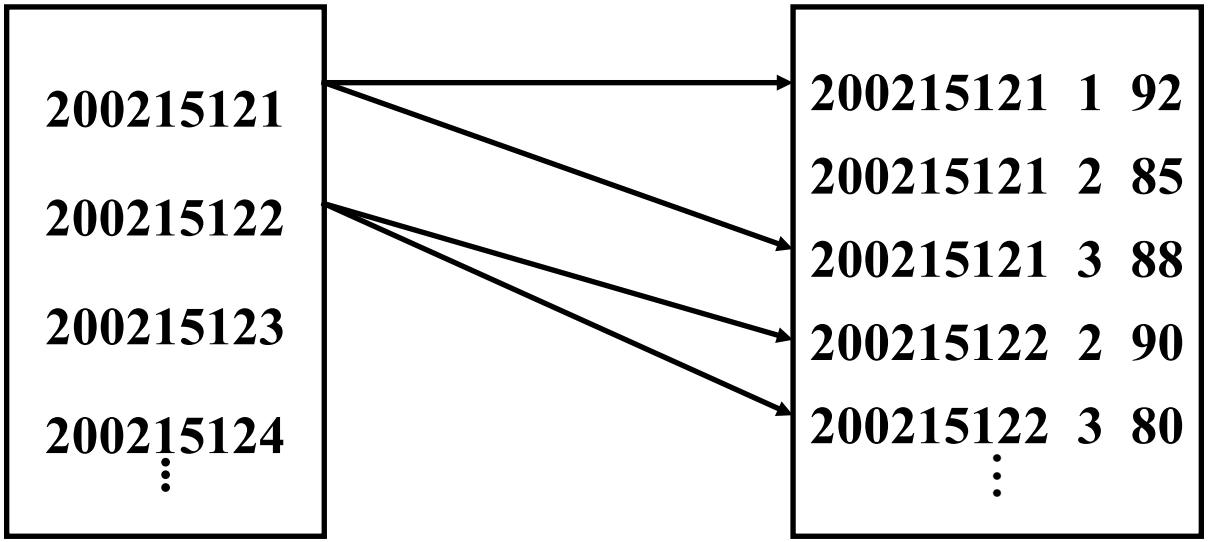
1. 嵌套循环方法 (nested loop)
   - 对外层循环 (Student) 的每一个元组 (s)，检索内层循环 (SC) 中的每一个元组 (sc)
   - 检查这两个元组在连接属性 (sno) 上是否相等
   - 如果满足连接条件，则串接后作为结果输出，直到外层循环表中的元组处理完为止

连接操作的实现（续）

2. 排序 - 合并方法 (sort-merge join 或 merge join)

- 如果连接的表没有排好序，先对 Student 表和 SC 表按连接属性 Sno 排序
- 取 Student 表中第一个 Sno，依次扫描 SC 表中具有相同 Sno 的元组
- 当扫描到 Sno 不相同的第一个 SC 元组时，返回 Student 表扫描它的下一个元组，再扫描 SC 表中具有相同 Sno 的元组，把它们连接起来
- 重复上述步骤直到 Student 表扫描完

连接操作的实现（续）



排序 - 合并连接方法示意图

连接操作的实现（续）

- Student 表和 SC 表都只要扫描一遍
- 如果 2 个表原来无序，执行时间要加上对两个表的排序时间
- 对于 2 个大表，先排序后使用 sort merge join 方法执行连接，总的时间一般仍会大大减少

连接操作的实现（续）

3. 索引连接 (index join) 方法

- 步骤：
  - ① 在 SC 表上建立属性 Sno 的索引，如果原来没有的话
  - ② 对 Student 中每一个元组，由 Sno 值通过 SC 的索引查找相应的 SC 元组
  - ③ 把这些 SC 元组和 Student 元组连接起来
  - 循环执行②③，直到 Student 表中的元组处理完为止

连接操作的实现（续）

4. Hash Join 方法

- 划分阶段 (building phase，也称为 partitioning phase)
  - 对包含较少元组的表 (比如 R) 进行一遍处理
  - 把它的元组按 hash 函数分散到 hash 表的桶中
- 试探阶段 (probing phase)： 也称为连接阶段 (join phase)
  - 对另一个表 (S) 进行一遍处理
  - 计算 S 元组连接属性的散列值，到相应的 hash 桶中去搜索匹配的 R 元组
  - 把该元组与桶中所有与之相匹配的 R 元组连接起来

连接操作的实现（续）

- 上面 hash join 算法前提：假设两个表中较小的表在第一阶段后可以完全放入内存的 hash 桶中
- 如果不能放入：多趟 hash join 算法

作业及实验

- 作业
  - P275-276　2（笔头）
  - P275-276　1,3,4（自己找到答案）
- 实验
  - P276 实验 9


2009/2/23


2009/2/23