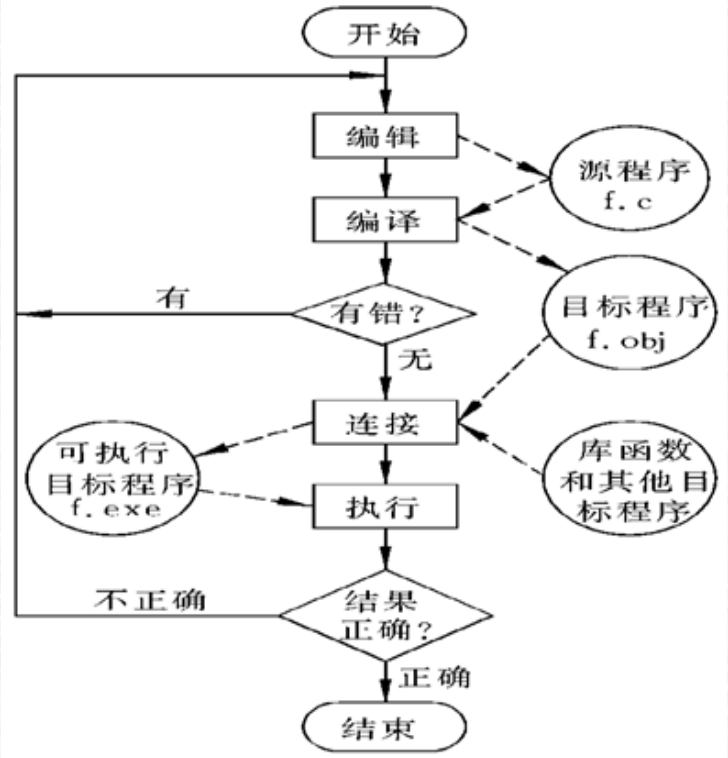
查询处理（基本操作的实现）

单 位：重庆大学计算机学院

程序语言的执行过程

- 一段C程序、Python程序是如何在计算机上执行的？



主要学习目标

- SQL查询过程
- 查询代价估算

思考问题（前测）

- SQL查询
  - 查找年龄大于20的学生姓名、年级
  - 查询“数据库”课程成绩大于80的学生姓名
-

一 概述

1. 查询处理过程

讨论1.RDBMS如何实现查询语句的执行？

检查语法和关系有效性 +
转换为关系代数操作操作

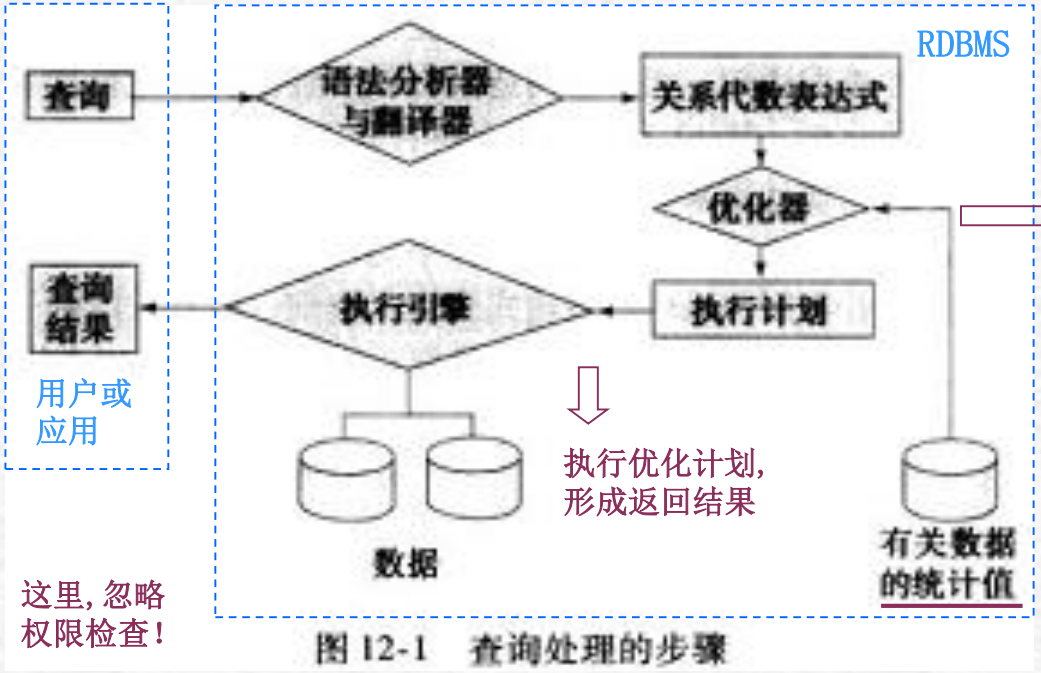


图 12-1 查询处理的步骤

1)查询处理包含哪些环节，及各环节的任务？

通过等价变换得到优化执行方案（操作执行次序）
含注释是否需要采用索引
具体采用的操作执行算法

$\sigma_{salary<75000}(\Pi_{salary}(instructor))$
$\Pi_{salary}(\sigma_{salary<75000}(instructor))$

2)什么是查询执行计划 p. 306?

指：执行一个查询的计算原语（标注了如何执行的一个或多个关系代数操作）的操作序列

2. 查询代价的度量

3)影响查询时间有哪些因素，及关键因素?

查询处理的代价可以通过该查询对各种资源的使用情况进行度量，这些资源包括磁盘存取、执行一个查询所用 CPU 时间，还有在并行/分布式数据库系统中的通信代价(我们将在第 18 - 19 章中讨论)。

然而在大型数据库系统中，在磁盘上存取数据的代价通常是最主要的代价，因为磁盘存取比内存操作速度慢。此外，CPU 速度的提升比磁盘速度的提升要快得多，这样很可能花费在磁盘存取上的时间仍将决定着整个查询执行的时间。一个任务消耗的 CPU 时间比较难以估计，因为它取决于执行代码的低层详细情况。尽管实际应用中查询优化器的确把 CPU 时间考虑在内，但为了简化起见我们将忽略 CPU 时间，而仅仅用磁盘存取代价来度量查询执行计划的代价。

我们用传送磁盘块数以及搜索磁盘次数来度量查询计算计划的代价。假设磁盘子系统传输一个块的数据平均消耗 t_T 秒，磁盘块平均访问时间(磁盘搜索时间加上旋转延迟)为 t_S 秒，则一次传输 b 个块以及执行 S 次磁盘搜索的操作将消耗 $b * t_T + S * t_S$ 秒。

二、 选择操作的实现

- [例1] Select * from student where <条件表达式> ；

  考虑<条件表达式>的几种情况：

  C1：无条件；

  C2：Sno＝’200215121’；

  C3：Sage>20；

  C4：Sdept＝’CS’ AND Sage>20；

选择操作的实现（续）

- 选择操作典型实现方法：
  - 1. 简单的全表扫描方法
    - ➢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
    - ➢适合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1-C2］　以C2为例，Sno＝‘200215121’，并且Sno上有索引(或Sno是散列码)
  - 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

- ［例1-C3］　以C3为例，Sage>20，并且Sage 上有B+树索引
  - 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1-C4]　以C4为例，Sdept＝‘CS’ AND Sage>20，如果Sdept和Sage上都有索引：
  - 算法一：分别用上面两种方法分别找到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
    - 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。

三、 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
- 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）

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

连接操作的实现（续）

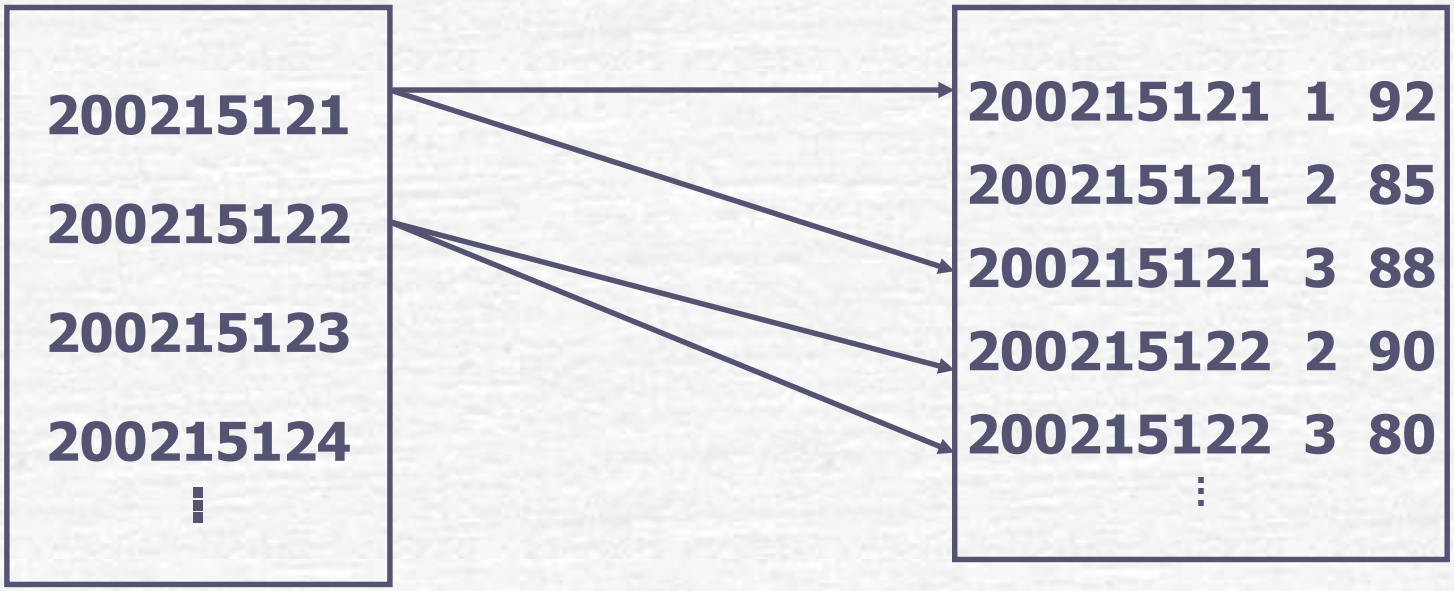
1. 嵌套循环方法(nested loop)
   - 对外层循环(Student)的每一个元组(s)，检索内层循环(SC)中的每一个元组(sc)
   - 检查这两个元组在连接属性(sno)上是否相等
   - 如果满足连接条件，则串接后作为结果输出，直到外层循环表中的元组处理完为止

连接操作的实现（续）

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

- 适合连接的诸表已经排好序的情况
- 排序－合并连接方法的步骤：
  - 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

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）

- 排序—合并连接方法的步骤（续）：
  - 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）

- 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，如果原来没有该索引

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

- 把连接属性作为hash码，用同一个hash函数把R和S中的元组散列到同一个hash文件中
- 步骤：
  - 划分阶段(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
    - 把S的元组散列到适当的hash桶中
    - 把元组与桶中所有来自R并与之相匹配的元组连接起来

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
- 以上的算法思想可以推广到更加一般的多个表的连接算法上

四 表达式计算(SQL查询)

讨论4. 表达式(复合操作)的计算如何执行？

1.(表达式计算中的)物化计算

1) 什么是物化，主要作用？

当采用物化方法时，我们从表达式的最底层的运算(在树的底部)开始。在该例子中，只有一个底层运算：*department* 上的选择运算。底层运算的输入是数据库中的关系。我们用前面研究过的算法执行这些运算，并将结果存储在临时关系中。在树的高一层中，使用这些临时关系来进行计算；这时的输入要么是临时关系，要么是来自数据库的关系。

高一层的运算 →

最底层的运算 →

物化-将中间计算结果作为临时关系存放，支持上一层计算！

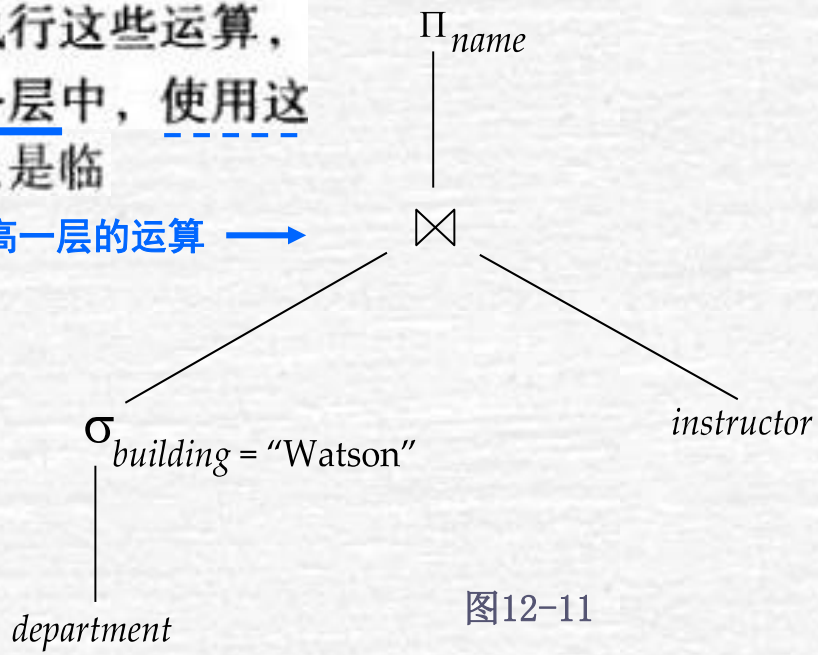


图12-11

2) 什么是流水线，主要作用？

3) 流水线的执行过程？

2. (表达式计算中的)流水线计算

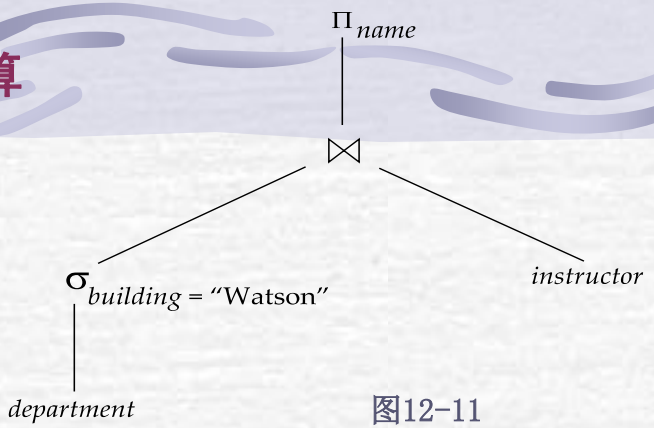


图12-11

通过减少查询执行中产生的临时文件数，可以提高查询执行的效率。减少临时文件数是通过将多个关系操作组合成一个操作的流水线来实现的，其中一个操作的结果将传送到下一个操作。这样的计算叫做**流水线计算**(pipelined evaluation)。

而不需要作为临时关系存储

例如，考虑表达式($\Pi_{a1,a2}(r \bowtie s)$)。如果采用物化方法，在执行中将创建存放连接结果的临时关系，然后在执行投影时又从连接结果中读入。这些操作可按如下方式组合：当连接操作产生一个结果元组时，该元组马上传送给投影操作去处理。通过将连接操作与投影操作组合起来，我们可以避免中间结果的创建，从而直接产生最终结果。

12.7.2.1 流水线的实现

通过将所有操作组合起来构成流水线，构造一个单独的复合操作，我们可以实现流水线。尽管对于各种频繁发生的情况，这个方法是可行的；但一般而言，希望在构造一个流水线时能重用各个操作的代码。

流水线作用：避免中间结果表的创建，提供执行效率！

在图 12-11 的例子中，三个操作均可放入一条流水线中；其中，选择操作的结果产生后传送给连接操作。接着，连接操作的结果产生后又传送给投影操作。由于一个操作的结果不必长时间保存，因此对内存的要求不高。然而，由于采用流水线，各个操作并非总是能立即获得输入来进行处理。

数据库的SQL引擎-OpenGauss

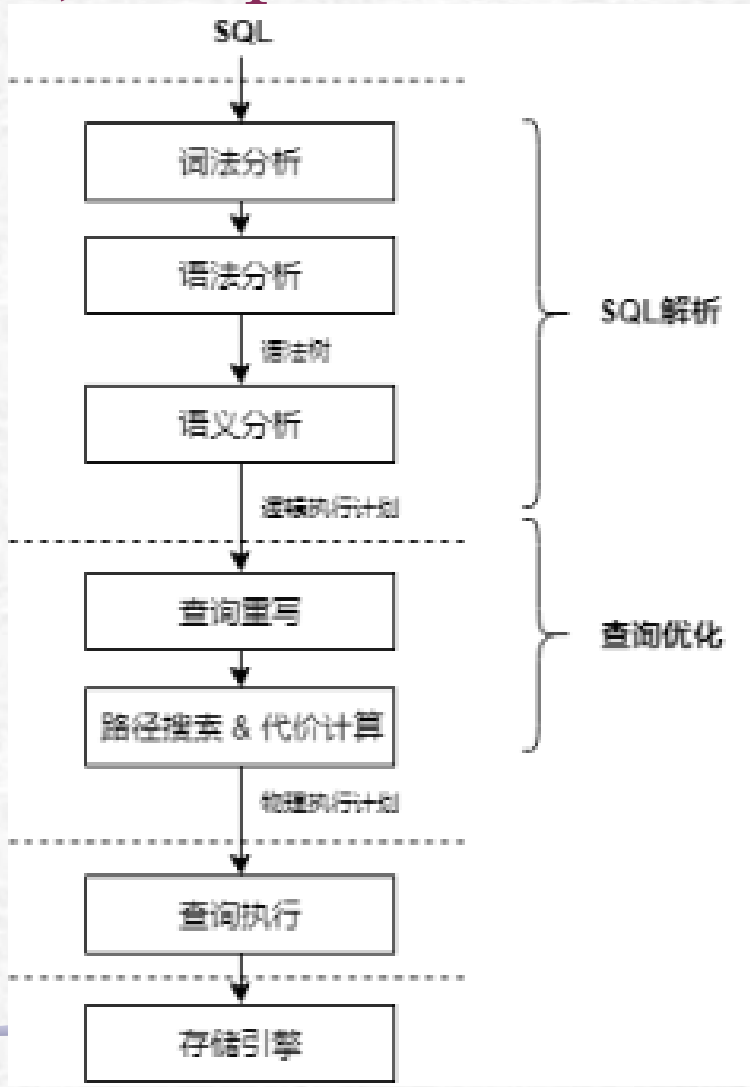
- 数据库的SQL引擎是数据库重要的子系统之一
- 它对上负责承接应用程序发送的SQL语句，对下负责指挥执行器运行执行计划。

数据库的SQL引擎-OpenGauss

- SQL引擎是数据库系统的重要组成部分。
- 主要职责是将应用程序输入的SQL语句在当前负载场景下生成高效的执行计划，在SQL语句的高效执行上扮演重要角色。

数据库的SQL引擎-OpenGauss

- 应用程序的SQL语句需要经过SQL解析生成逻辑执行计划、经过查询优化生成物理执行计划，然后将物理执行计划转交给查询执行引擎做物理算子的执行操作



数据库的SQL引擎-OpenGauss

- SQL解析通常包含词法分析、语法分析、语义分析几个子模块。
- SQL是介于关系演算和关系代数之间的一种描述性语言，它吸取了关系代数中一部分逻辑算子的描述，而放弃了关系代数中“过程化”的部分。
- SQL解析主要的作用就是将一个SQL语句编译成为一个由关系算子组成的逻辑执行计划

数据库的SQL引擎-OpenGauss

- 词法分析:从查询语句中识别出系统支持的关键字、标识符、运算符、终结符等，确定每个词固有的词性。
- 语法分析:根据SQL的标准定义语法规则，使用词法分析中产生的词去匹配语法规则，如果一个 SQL 语句能够匹配一个语法规则，则生成对应的抽象语法树（Abstract Syntax Tree，AST）。
- 语义分析:对语法树进行有效性检查，检查语法树中对应的表、列、函数、表达式是否有对应的元数据，将抽象语法树转换为逻辑执行计划(关系代数表达式)。

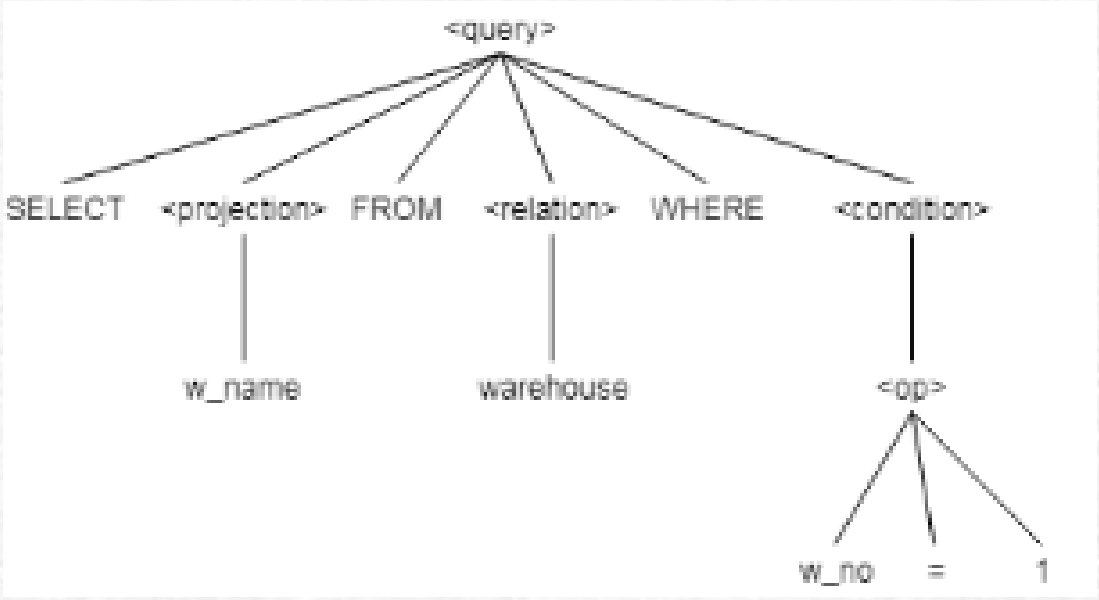
数据库的SQL引擎-OpenGauss

- SELECT w_name FROM warehouse WHERE w_no = 1;
- 词法分析：该SQL语句可以划分的关键字、标识符、运算符、常量等原子单位如表1所示

词性	内容
关键字	SELECT、FROM、WHERE
标识符	w_name、warehouse、w_no
操作符	=
常量	1

数据库的SQL引擎-OpenGauss

- 语法分析会根据词法分析获得的词来匹配语法规则，最终生成一个抽象语法树，每个词作为语法树的叶子节点出现，如下图所示



数据库的SQL引擎-OpenGauss

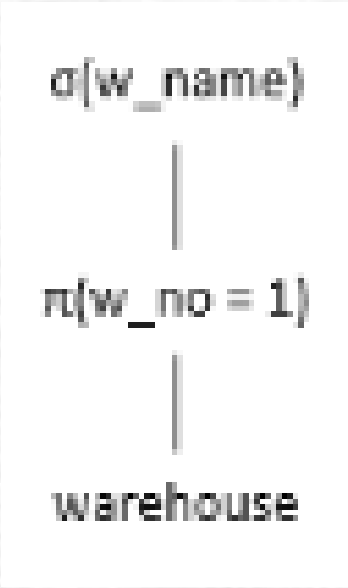
- 抽象语法树表达的语义还仅仅限制在能够保证应用的SQL语句符合SQL标准的规范，但是对于SQL语句的内在含义还需要做有效性检查。

(1)检查关系的使用:FROM 子句中出现的关系必须是该查询对应模式中的关系或视图。

(2)检查与解析属性的使用:在SELECT语句中或者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。

(3)检查数据类型:所有属性的数据类型必须是匹配的。在有效性检查的同时，语义分析的过程还是有效性语义绑定(Bind)的过程，通过语义分析的检查，抽象语法树就转换成一个逻辑执行计划。逻辑执行计划可以通过关系代数表达式的形式来表现。

数据库的SQL引擎-OpenGauss



数据库的SQL引擎-OpenGauss

- 查询重写则是在逻辑执行计划的基础上进行等价的关系代数变换，这种优化也可以称为代数优化。
- 虽然两个关系代数式获得的结果完全相同，但是它们的执行代价却可能有很大的差异，这就构成了查询重写优化的基础。

随堂小测试

- 一条SQL语句要经历几个步骤才能被数据库执行？
- 两个表的连接操作有几种实现方法？

课堂小结和课后作业安排

- 基本知识：
  - 查询处理过程
  - 选择实现的方法
  - 连接实现的方法
- 延展性学习：
  - 如何设计一个优秀的查询处理方案。
- 作业

  第12章习题(中文6版)：12.2, 12.3 b), 12.6 a).
- 预习
  - 第15讲-查询处理（查询优化）（课前预习资料）
  - 简略介绍下一讲的学习内容

下一讲的学习内容

核心任务：代数优化！

- 查询优化过程
- 关系表达式的优化（代数优化）
- 执行计划的优化（物理优化）