# 数据库系统实现第五次作业报告

## 一．功能介绍

1. 实现的功能：数据表的增删改查；索引创建；多表连接；基于统计信息的查询优化；

2. 所支持的 SQL 语法

- 创建数据库

  Create database mydb;

- 创建表，本实验中创建了三个表

  Create table supplier(S_superkey i, S_name c30, S_address c40, S_nationkey i,S_phone c20,S_acctbal f,S_comment c110);

  Create table partsupp(PS_PARTKEY i, S_superkey i,PS_AVAILQTY i,PS_SUPPLYCOST i,PS_COMMENT c240);

  Create table part(PS_PARTKEY i,P_NAME c55,P_MFGR c25,P_BRAND c10,P_TYPE c25,P_SIZE i,P_CONTAINER c10,P_RETAILPRICE f,P_COMMENT c23);

- 插入一条数据

  insert into supplier values(0, "taoyouxian", "beijing", 110,"110",2.333,"hello word");

- 更新数据内容

  Update supplier set S_name = "zhangheng" where S_superkey = 1;

- 删除内容

  Delete from supplier where S_name = 'zhangheng';

- 查询数据

  select *

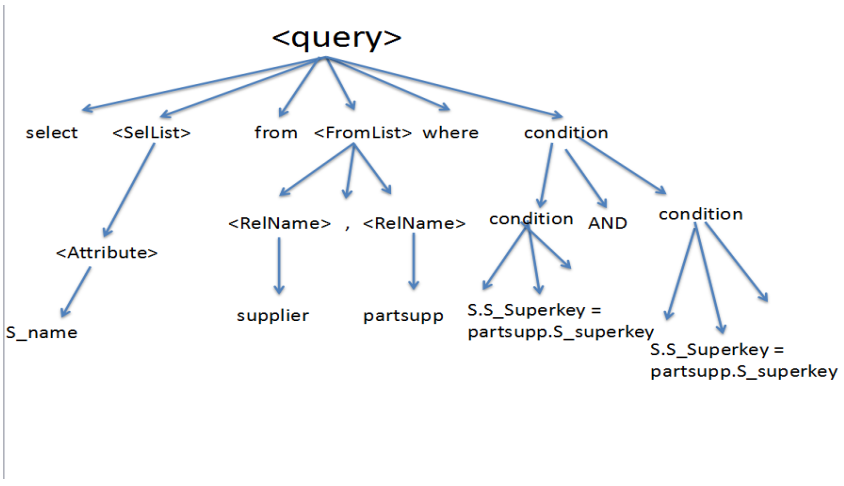
  from supplier

  where supplier.S_name = 'zhangheng';

3 语法树的形式，并举例说明

以如下语句为例

```
select S_name
from supplier,partsupp
where supplier.S_superkey = partsupp.S_superkey and S_superkey = 2;
```



4 查询优化划的形式，并举例说明

查询优化又分为代数优化和物理优化，由查询优化器将查询预处理器所生产成的关系代数初始查询树转化为一个预期所需执行时间较小的等价的关系代数查询树，得到一个可以被转换成最有效的物理查询计划的一个“优化”的逻辑查询计划。

物理优化是从一个查询计划产生物理查询优化时，为逻辑查询计划的每一个操作符选择实现算法并选择这些操作符的执行顺序，还包括许多实现的细节，包括被查询的关系是怎样被访问的，即获得所存储数据的方式以及一个关系合适或是否应当被排序等。必须估计每个可能选项的执行代价，使用代价估计来评价一个查询计划。

5 查询优化方法：条件下推；通过基于代价估算的表连接顺序调整；

一个查询的实际执行所需的代价包括：

- 磁盘访问代价
- 存储代价
- 计算代价
- 内存使用代价，等

6 查询执行策略：查询优化的方法主要通过进行统计信息的分析，以及代价的估算进行查询计划的选择；

7 数据字典：数据字典是指对数据的数据项、数据结构、数据流、数据存储、处理逻辑、外部实体等进行定义和描述，其目的是对数据流程图中的各个元素做出详细的说明，使用数据字典为简单的建模项目；

8 记录和维护统计信息的方法：通过创建一个数据结构来保存数据的统计信息；

# 二 一些模块的实现

1 重要数据结构

```
//属性的统计信息
typedef struct attrStat{
   float numTuples; // num distinct values for this attribute
   float maxValue; // maxValue for attribute
   float minValue; // minValue for attribute
} attrStat;

// 进行 join 操作过程中的统计信息
typedef struct costElem{
   int joins;              // bitmap of rels in (S-a)
   int newRelIndex;    // index of a
   float numTuples;    // number of tuples of
   float cost;             // cost of joining (S-a) with a to get S
   int indexAttr;        // index attribute. is -1 if no index is used
   int indexCond;        // index condition. is -1 if no index is used
   std::map<int, attrStat> attrs;    // map of attribute statistics
} costElem;
```

2 函数调用图

RC QO_Manager::Compute(QO_Rel *relOrder, float &costEst, float &tupleEst)

CalculateOptJoin(newRelsInJoin, i)

RC QO_Manager::CalculateJoin(int relsInJoin,

int newRel, int relSize, float &cost, float &totalTuples, map<int, attrStat> &attrStats,

int &indexAttr, int &indexCond)

3 提供给其他模块的接口函数及主要内部函数描述：

对其他模块提供的接口函数，将查询表达式的相关信息传入该函数；

QO_Manager::QO_Manager(QL_Manager &qlm, int nRelations, RelCatEntry *relations, int nAttributes, AttrCatEntry *attributes,

int nConditions, const Condition conditions[]) : qlm(qlm)；

析构函数，评估完毕后释放所分配的空间

```
QO_Manager::~QO_Manager(){
  for(int i=0; i < nRels; i++){
    map<int, costElem*>::iterator it;
    for(it= optcost[i].begin(); it != optcost[i].end(); ++it){
      delete it->second;
    }}}
```

打印在优化过程中的统计信息；

```
RC QO_Manager::PrintRels(){
```

进行执行计划的代价评估；

RC QO_Manager::Compute(QO_Rel *relOrder, float &costEst, float &tupleEst)；

CalculateOptJoin(newRelsInJoin, i)

RC QO_Manager::CalculateJoin(int relsInJoin,

int newRel, int relSize, float &cost, float &totalTuples, map<int, attrStat> &attrStats,

int &indexAttr, int &indexCond)；