**小型数据库搭建——数据结构实习报告**

211294029 李梦麟 211294043 江炫烨

0 摘要

本次实习作业希望利用C++将数据结构课程中所学知识应用于小型数据库的构建中，通过实践加深对数据逻辑结构与其所映射的物理结构的理解，根据需求情境与数据结构特点，采用不同类型的数据结构解决问题。实习作业中主要应用B+树、字符串、数组等数据结构来实现关系型数据库中不同功能的类的定义与具体操作，包括数据存储、顺序访问、插入、删除、查找、范围查询、SQL语句解析等操作。由于尚未实现导入外部数据的功能，本次实习将具体的业务定位为小型企业的员工管理。

1 实践背景

在信息资源管理发展的初级阶段，数据存储和数据管理通过传统的文件系统完成，随着信息技术发展，数据生产与日俱增，计算机开始应用于数据管理，数据库技术作为现代化的信息管理技术，能够很好地应对当下数据量大、信息繁杂等问题，能够提高数据管理的效率和准确性、降低数据冗余，节约存储空间，同时提供多种查询方式，方便用户进行数据筛选和分析，增强了数据的安全性和可靠性。

数据库是结构化信息或数据的有序集合，一般以电子形式存储在计算机系统中。通常由数据库管理系统(DBMS)来控制。为提高数据处理与查询效率，当下最常见的数据库通常以行和列的形式将数据存储在一系列的表中，以支持用户便捷访问、管理、修改、更新、控制和组织数据。目前大多数数据库使用结构化查询语言(SQL)来查询、操作和定义数据，用户只需在高层数据结构上进行数据处理，无须用户指定数据的存取方法,也无需用户了解具体的数据存储方式,就可以使用SQL对不同数据库进行数据访问控制。数据库按其结构可分为多种类型，包含层次数据库、网状数据库、关系数据库、图数据库等，其中在关系型数据库里，项被组织为一组具有列和行的表，实现结构化数据的存储，使其访问更加便捷灵活。

关系型数据库广泛应用于企业的人员、交易等管理系统上。于企业而言，建立高效的信息管理系统是其适应社会发展形势的必经之路，其中面向小型企业的员工管理系统，要求整体性强、操作方便、流程简洁，利用关系型数据库进行数据存储以及通过SQL进行信息的录入、更新、删除、查找即可较好地满足这一要求，因此本次实习旨在实现关系型数据库基本架构搭建的基础上，进一步实现员工信息基本的增删改查等功能。

2 理论基础

2.1 关系型数据库基本架构

2.2 几类数据结构

B+树：特点+功能对应

串：

数组：

3 实践内容

本次实习作业采用C++为主要编程语言，C++17及以上的编译环境，利用C++面向对象的特性构建了BPlusTree（B+树类）、Table（实现数据存储和操作的核心类）、Colunm（存储数据库关系模式的类）、Row（存储数据行）、Page（对应B+树的节点）、DB（进行输入交互与语句解析的虚拟机）等类来实现数据库的功能。接下来将详细介绍各个类的构建思路。

3.1 类构建

1. BPlusTree类

BPlusTree类主要实现了B+树的定义及相关操作，B+树是一种常用的自平衡搜索树，广泛应用于数据库和文件系统等领域。它是基于B树的一种变体，相比于B树，B+树在内部节点不存储数据，只存储键值（或范围），而所有的数据都存储在叶子节点中。这种设计使得B+树具有更高的查询效率和更好的顺序访问特性。

下面介绍B+树的主要方法的实现逻辑：

1. 插入（Insert）：

- 从根节点开始，根据插入键值找到叶子节点。

- 如果叶子节点未满，直接插入键值并保持有序。

- 如果叶子节点已满，进行分裂操作：

- 将叶子节点一分为二，分成左右两个节点。

- 将中间键值提升到父节点，并递归地进行插入操作。

- 如果父节点已满，继续向上递归进行分裂操作，直到根节点。

2. 删除（Delete）：

- 从根节点开始，根据删除键值找到叶子节点。

- 如果叶子节点中存在要删除的键值，直接删除。

- 如果叶子节点删除后过小，进行合并或者借用操作：

- 如果兄弟节点有富余的键值，可以借用一个键值过来。

- 如果兄弟节点的键值不够，可以进行合并操作，将当前节点和兄弟节点合并成一个节点。

- 如果父节点删除后过小，继续向上递归进行合并或者借用操作，直到根节点。

3. 查找（Search）：

- 从根节点开始，根据查找键值逐层向下搜索。

- 如果当前节点是内部节点，根据键值选择下一个子节点。

- 如果当前节点是叶子节点，根据键值在叶子节点中进行查找。

4. 范围查询：

- B+树天然支持范围查询，可以通过最左叶子节点和最右叶子节点进行范围限制，减少不必要的遍历。

总结起来，B+树的主要方法包括插入、删除、查找和范围查询。通过合适的平衡操作，B+树能够在动态数据集中高效地支持插入、删除和查找操作，使其成为一种广泛应用于存储和检索大量数据的数据结构。

如下图1展示了BPlusTree的类图，其中MTreeNode结构体的定义如下：

template <class T>

struct MtreeNode {

        bool isLeaf;

        int n;

        MtreeNode\* parent;

        std::vector<T> keys;

        std::vector<MtreeNode\*> pointers;

        std::vector<int\*> recptrs;

    };

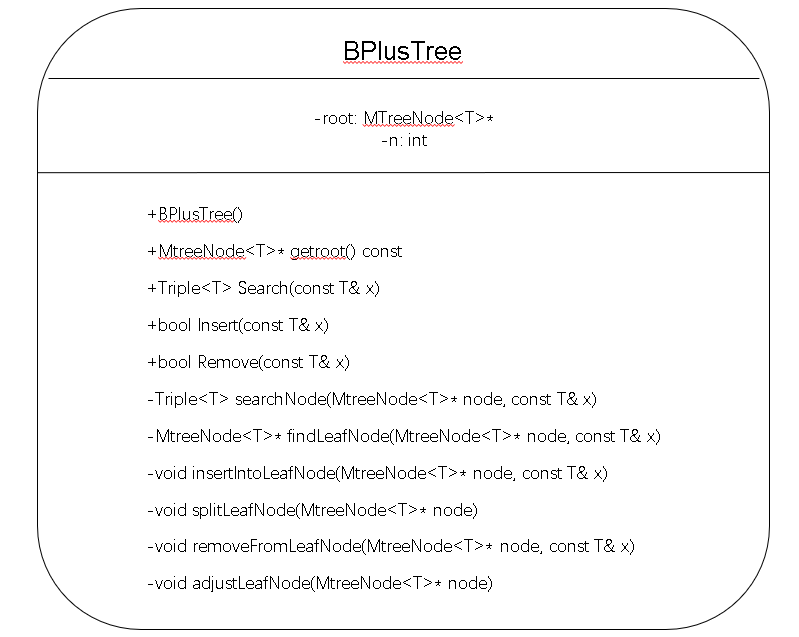


图1

1. Row、Column和Page类

Row类和Column类是为了方便实现存储数据库关系模式而设计的类。Row类主要用于存储数据行，其使用的核心数据结构是一个map(即键值对)，用来存储列名和相对应的值。为了保证Row可以存储字符串、整形、浮点型等多种类型的数据使用了一个特殊的数据结构即std::variant类型，std::variant 是 C++17 引入的标准库类型，它提供了一种能够容纳不同类型值的数据结构，类似于联合体（union）或变体（variant）的概念。不同于联合体，std::variant 保证了类型安全和异常安全，并提供了更多的功能和灵活性。

Column类主要储存数据表的定义，其主要的数据成员是用std::string类型存储的列的列名及其对应的数据类型。通过这两个类实现了数据表结构的控制。

而Page类则实现了一个分页，其中存储的是一个Row类的数组。设置该类是为了实现数据的存储结构与B+树类的连接，即Page类实际上就是B+树叶子节点下指针指向的数据。同时为了能够正常使用B+树相关的插入删除操作，还对该类进行了>、<、==等操作符的重载。下面的图2、图3、图4展示了Row和Column对应的类图。

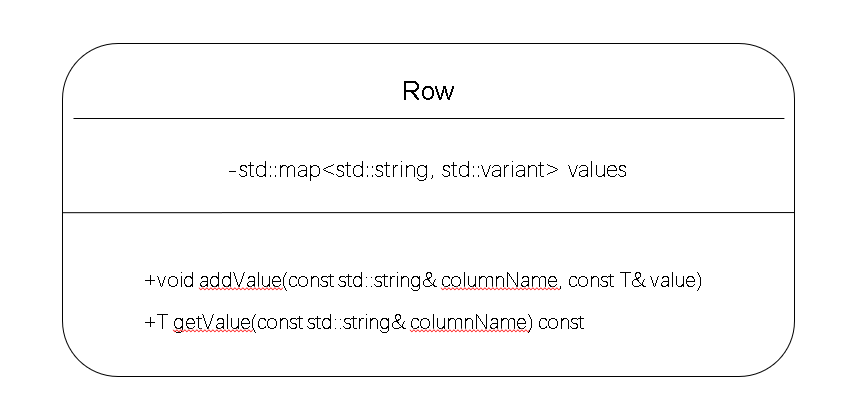


图2

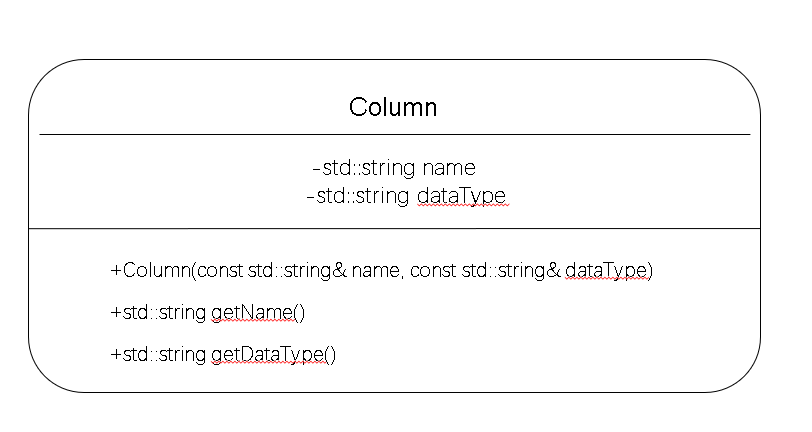


图3

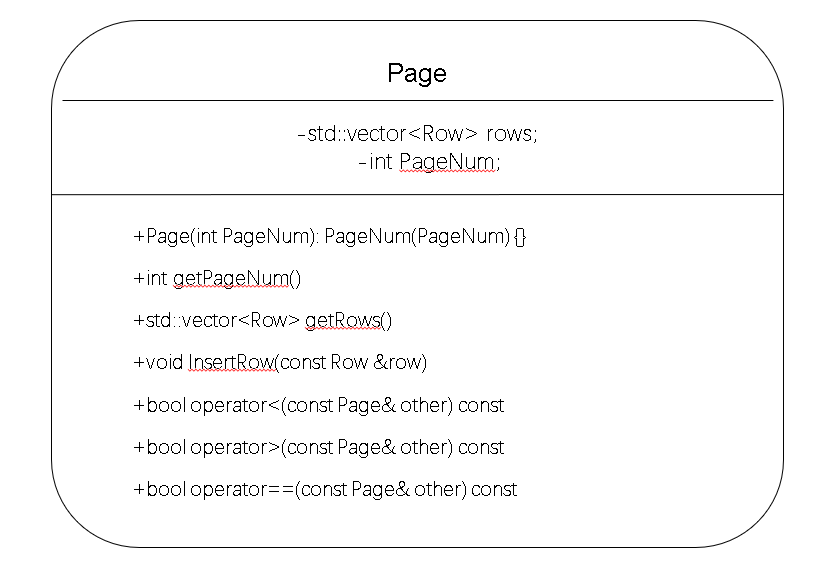


图4

1. Table类

Table类即对应数据库中的Table，其拥有如下数据成员：

    std::string name;               // 表名

    std::vector<Column> columns;    // 列列表

    BPlusTree<Page> pages;          // 数据存储结构

    uint32\_t nrow;                  // 表存储数据的行数

Table通过columns来控制输入数据格式的正确、通过调用B+树相应的插入、删除、搜索等方法实现数据的插入、删除等操作。

下图为Table类对应的类图

图形用户界面, 文本, 应用程序

中度可信度描述已自动生成

图5

1. DB类

DB类即database类，DB类只拥有一个数据成员，一个Table类的数组来存储创建的数据表。DB类实现了这样一些功能：

-打印一个database>的提示符

-解析例如”.exit”（退出数据库）等的元命令

-解析SQL语句

-执行SQL操作

其中实现解析SQL语句的过程中采用了许多字符串相关的数据结构的知识，包括子串定位、字符串的分割等。实习作业中简单实现了KMP算法，但因为SQL语句的语法结构较为固定，实际实现的过程中采用的是C++中自带的find()函数。执行SQL操作的过程被分割为多个子函数，其实现的逻辑主要是通过读入SQL语句解析的结果并调用Table类的相应方法来实现SQL操作。其中SQL语句解析结果的读入主要是通过一个Statement类，其具体定义如下：

class Statement {

    //定义SQL语句类型

    public:

        StatementofSQL type;

        std::vector<std::string> create\_request;

        std::vector<std::string> insert\_request;

        std::vector<std::string> select\_request;

        std::vector<std::string> update\_request;

        std::vector<std::string> delete\_request;

};

3.2 操作示例

4 总结与局限

通过本次实习任务，加深了对B+树、字符串、数组等数据结构的理解，通过C++面向对象的特性实现了数据库中几种功能的类的构建，在实践中学会根据任务需求选择合适的解决方案，再结合数据结构的类型特点选择最适于某情境的数据结构与算法，如....

出现的问题与应对方案?

优化？

5 附录代码