数据库系统原理与实现

姓名：洪宇
学号：2022080901022

实验三

实验内容

project 3 的内容为：

1. 读取一个 CSV 文件，其中每行代表一个对象，各个字段由逗号分隔。第一个字段为一个无符号整数，相当于该对象的主键（此处称为 id）；其余字段为该对象的属性。总字段数不固定，且除要求第一个字段为无符号整数外，其余字段类型不定。
2. 给定一个属性值 A，可以获得含有 A 的对象的集合。
3. 给定一个对象 id，可以获得该对象的所有属性。

由上述定义可知，该 CSV 文件并不能对应到一张关系型数据库的表。这是因为，每个对象具有的属性数是不定的，同时各属性的类型也不定，因此没有“列”，自然也没有表的元数据。

Bitmap

首先，为什么会有 bitmap 这种数据结构呢?

在对数据库的查询中，经常出现需要根据给定的字段值查询具有该字段值的对象集合的情况，且该字段并不一定是主键。如果查询的不是主键，则无法利用B+树等数据结构加速查找，而只能遍历所有叶节点。当然，可以给特定的“列”创建辅助索引，也即以这些“列”上的值为键值对中的键、对应记录的主键的值为键值对中的值来创建一颗B+树，进而在该B+树中查询记录主键的集合。但如此实现仍有两个问题：

- 必须要预先创建辅助索引。假如要查询的“列”没有建辅助索引，则仍然需要遍历叶节点。
- 辅助索引将占据更多空间。

而在现实场景中，经常会出现某种属性值在许多对象中出现的情况。比如，如果有一个存储学生信息的表，其中有一个字段表示性别，则其值只有“男”、“女”两种。对于这种可能的值不多、而每种值可以映射到的对象很多的情况，便可以用 bitmap 来加速查询。一个典型的例子如下图：

Colleagues		
name	street	city
peter	unistreet	new york
steve	macstreet	cuppertino
mike	longstreet	berlin
tim	unistreet	berlin
hans	msstreet	new york
jens	longstreet	cuppertino
frank	unistreet	new york
olaf	macstreet	berlin
stefan	longstreet	berlin
alekh	unistreet	berlin
felix	macstreet	new york
jorge	longstreet	berlin

city		
new york	cuppertino	berlin
1	0	0
0	1	0
0	0	1
0	0	1
1	0	0
0	1	0
1	0	0
0	0	1
0	0	1
0	0	1
1	0	0
0	0	1

street			
unistreet	macstreet	longstreet	msstreet
1	0	0	0
0	1	0	0
0	0	1	0
1	0	0	0
0	0	0	1
0	0	1	0
1	0	0	0
0	1	0	0
0	0	1	0
1	0	0	0
0	1	0	0
0	0	1	0

以列“city”为例，为它的各个可能值建一个 bitlist，bitlist 的总长度与总对象数相等。如果某一对象有该值，则对应位置记为1，否则为0。有了 bitmap，不仅能够给定一个属性值进而快速给出对象集合，还能很容易地进行组合查询。比如，假如要查询同时具有“new york”和“unistreet”属性的对象，只需取两个 bitlist 的并集即可。

Roaring Bitmap

然而，尽管 bitmap 能够进行快速查找及方便的集合操作，但如果这些 bitlist 构成的矩阵比较稀疏的话（1多0少），则空间效率非常低下——这就需要引入 bitmap compression。本实验采用的是 Roaring Bitmap。

基本原理

Roaring Bitmap 将整数集合划分为 2^{16}（65536）个整数的块，每个块存储一个特定范围内的整数。每个块可以使用不同的容器（即压缩方式），以适应数据的稠密度和稀疏度。容器一般为下面三种：

1. `Array`：
    - 适用于稀疏数据。
    - 使用一个数组来存储整数。
    - 当整数的数量较少时（通常小于 4096）使用。
2. `Bitmap`：
    - 适用于稠密数据。
    - 使用 bitmap 来存储整数。
    - 当整数的数量较多时（通常大于等于 4096）使用。
3. `Run`：
    - 适用于连续的整数序列。
    - 使用 RLE（Run-Length Encoding）来存储整数。

- 当整数序列连续时使用。

由上面的介绍可知，Roaring Bitmap 会将一个32位整数分为高16位和低16位两部分，高16位用于分块，各个块的存储所用的数据结构不固定（如数组、树等）；低16位则根据稀疏程度来确定容器类型。比如，当集合内处于 $[0, 2^{16}]$ 内的整数只有1、2两个数字时，就用 `Array` 来存储；而当该范围内的数字多到超过一定界限时，便可以将容器调整为 `Bitmap` 。

实现

在 project 3 中，Roaring Bitmap 采用的是 CRoaring 库。实际上只需要明确几点：

1. Roaring Bitmap 存储的是32位无符号整数（CRoaring 也给出了64位的实现，不过本实验仍使用32位）
2. 存进 bitmap 的整数实际上就是每个对象的 id
3. 为了区分不同的 bitmap 对应了什么属性，需要建立属性名到某一 bitmap 的映射。

设计

源代码中， `Bitmap` 类的核心成员只有2个：

- `std::unordered_map<string, Roaring> tagMaps_`

  建立属性名到一个 Roaring Bitmap 的映射

- `std::unordered_map<unsigned int, std::vector<string>> records_`

  建立对象 id 到其具有的各个属性的映射

文件

读取文件

思路很简单，打开文件后逐行读入，对于每行：

1. 利用 `std::stringstream` 和 `getline` 将该行按逗号分隔，并把每个分隔后的元素添加到数组 `row` 中；
2. 将 `row` 的第一个元素对应的整数值作为键、其余元素构成的数组作为值，添加到 `records_` 中；
3. 对于 `row` 中除第一个元素外的所有元素（即该对象的属性），在 `tagMaps_` 中查询是否有该属性名对应的键：
   i. 若没有，则用 `tagMaps_.insert({ row[i], roaring::Roaring() });` 插入一个新的从属性名到 Roaring Bitmap 对象的映射；
   ii. 否则，直接用 `tagMaps_[row[i]].add(std::stoi(row[0]));` 将该对象的 id 加入到该属性名对应的 Roaring Bitmap 中。

写回文件

当程序终止时需要写回文件，即遍历 `records_` 的所有键值对，每个键值对写到文件中的一行；先写入键（即对象 id），再以逗号作为分隔，写入对应集合中的各个值，最后以换行符结束。

基本操作

插入对象

`insert` 接受对象属性值构成的数组及其 id。首先要判断该对象 id 是否已存在：

```
if (records_.find(id) != records_.end()) return false;
```

如果 id 不存在，除了要插入 `records_` 外，还需要维护 `tagMaps_`；后者与“读取文件”的第三步基本相同。其中，`tagMaps_[tag]` 是一个 Roaring Bitmap 对象，在该对象上调用 `add` 便是把一个整数加入到了 bitmap 中，接着 CRoaring 会在内部处理分块、容器选择等操作。

查询对象

`contain` 提供了一个给定属性值 `tag`，查询某一 `id` 是否具有该属性的接口。这可以分为两步：

1. 查询是否具有该属性值：`if (tagMaps_.find(tag) == tagMaps_.end()) return false;`
2. 调用 CRoaring 的 `contains` 函数，判断某一整数是否在集合中：`tagMaps_[tag].contains(id);`

给定属性，查询 id

仍分为两步：

1. 查询是否具有该属性值
2. 将获得的 id 集合放入一个 `vector` 中并返回

第二步实际上涉及到将 Roaring Bitmap 存储的压缩后的 bitmap 解压为整数集，这可以通过 CRoaring 的接口 `toUint32Array` 来实现。需要注意的是，CRoaring 提供的 C++ 接口实际上只是对其 C 实现的很“薄”的包装，这些 API 都没有提供针对 C++ 特有类型的额外支持——也就是说，我在传入一个 `vector` 的首地址前，必须要初始化它的大小，因为 CRoaring 不会在内部动态调整这个 `vector` 的大小。因此，在调用 `toUint32Array` 前，必须先初始化 `vector` 的元素个数，如：

```
std::vector<unsigned int> ret(tagMaps_[tag].cardinality());
```

其中 `tagMaps_[tag].cardinality()` 即该属性对应的整数集合中，存储的（不重复的）整数个数。

这又是一个略 tricky 的地方，为之后的单元测试带来了不便。这里先埋个伏笔。

给定 id，查询属性

实际上就是以 id 为键，查询 `records_` 中是否有对应值，在此不必详述。

集合操作

源代码中实现了求两个 bitmap 交集、并集、差集的接口，它们在结构上都是很类似的，因此这里一起解释。

首先需要判断传入的两个属性值是否在 `tagMaps_` 中存在，不存在则直接返回空集（空 `vector`）；接着判断两个属性值是否相等，若相等，则：

- 交集：直接返回 `tag1` 对应的整数集
- 并集：直接返回 `tag1` 对应的整数集
- 差集：返回空集

从 bitmap 到整数集的过程与“给定属性，查询 id”部分的操作类似，均需要调用 `toUint32Array`，且均需要提前初始化 `vector` 的元素个数。这里可以直接 回收伏笔：由于不能确定集合操作的结果会包含多少元素，因此**返回的 `vector` 的末尾很可能有许多0元素**。然而，GoogleTest 提供的宏会认为两个大小不等的 `vector` 不等，进而测试失败。解决方法将在后文的单元测试部分说明。

另外值得说明的是：当集合操作结果为空集时，CRoaring 并不会将传入的指针所指向的数组变为空数组。为了与查询的属性不存在于 `tagMaps_` 中时直接返回 `{}` 以表示空集的逻辑一致，需要作下面的处理：

```
if (std::all_of(ret.begin(), ret.end(), [](int i) { return i == 0; }))
    return {};
else
    return ret;
```

即判断集合操作结果是否为一个全为0的数组，如果是的话返回 `{}`。

测试

单元测试使用的是 GoogleTest（简称 GTest），其与 Catch2 最大的区别在于：**各个 `TEST` 之间是完全独立的**。而且，在 CLion 中，GTest 的各个 Test 可以直接运行，不用像 Catch2 那样打标签，真的好用太多了。

文件

读取文件

读取文件的测试在 `TEST(BitmapTest, ReadFile)` 中。因为不应该依赖于其他模块，所以直接用 `fstream` 写入文件，随后调用 `readFile` 并分别测试 `records_` 和 `tagMaps_` 成员是否都与写入文件的内容相符。

比如，通过以下代码来写入同时具有属性“Aliens”和“Mars”的对象：

```
for (int i = 1; i < GAP + 1; ++i)
    fout << i << "," << "Aliens" << "," << "Mars" << "\n";
```

接着，测试对于这些 id，是否具有这些属性：

```
std::vector<std::string> vec { "Aliens", "Mars" };
    for (int i = 1; i < GAP + 1; ++i)
        EXPECT_EQ(bitmap.records_[i], vec) << i;
```

`<< i` 用于调试，是 GTest 一个很方便的功能：当某次循环中 `EXPECT_EQ` 失败时，便会打印该次循环对应的 `i` 值。

再验证该 bitmap 的基数：

```
EXPECT_EQ(bitmap.tagMaps_["Mars"].cardinality(), GAP);
```

最后，测试对于这些属性，是否正确对应了原先插入的 id：

```
for (int i = 1; i < GAP + 1; ++i)
    EXPECT_EQ(bitmap.tagMaps_["Mars"].contains(i), true);
```

写回文件

写文件实际上只需要用到 `records_`，于是先手动为 `records_` 赋值，如：

```
bitmap.records_[1] = { "A", "Atlantic" };
```

接着调用 `writeFile` 写入文件，随后用 `fstream` 打开文件，并逐行检查正确性，用到的仍是 `stringstream` + `getline` 的技巧。随后需检查：

1. 读到的该对象 id 存在于 `records_` 中
2. 读到的属性值与 `records_` 中该 id 的属性值相符

```
// 检查 id
EXPECT_NE(bitmap.records_.find(stoi(row[0])), bitmap.records_.end());

// 检查属性
EXPECT_EQ(equal(vec.begin(), vec.end(),
          row.begin() + 1, row.end()), true);
```

其余接口

`TEST(BitmapTest, API)` 中的测试用例非常容易理解，并且我也给出了较为详细的注释，因而在此不再逐步分析。用例的测试范围包含了：

- 插入对象成功
- bitmap 个数正确
- 每个 bitmap 的基数正确
- 每个 bitmap 存储的 id 正确
- 查询的属性不存在时返回 `false`
- 查询的属性存在，但所查 id 不在该 bitmap 中时返回 `false`
- 查询的属性存在，且所查 id 在该 bitmap 中时返回 `true`
- 查询的属性不存在时，返回的 id 集合为空集
- 查询的属性存在时，返回的 id 集合正确
- 求并、交、差集的两个属性中，若有至少一个属性不存在，则返回空集
- 一个属性与自身的并集还是自身
- 一个属性与自身的交集还是自身
- 一个属性与自身的差集返回空集
- 两个集合的并集返回的 id 集合正确
- 两个集合的交集返回的 id 集合正确
- 两个集合的差集返回的 id 集合正确
- 给定 id，查询的属性值正确

这里要说明一下解决“`vector` 长度不等时 GTest 会直接判断不等”问题的方法：

```
auto diff = bitmap.getDifference("Forest", "Faun");

EXPECT_TRUE(equal(diff.begin(),
    diff.begin() + min(diff.size(), ids.size()), ids.begin()));
```

即利用 `equal` 来比较 `diff` 和 `ids` 前 `min(diff.size(), ids.size())` 部分的元素是否相等。

测试结果

```
/Users/yu/CLionProjects/bitmap/cmake-build-debug/tests/bitmap_tests
[==========] Running 3 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 3 tests from BitmapTest
[ RUN      ] BitmapTest.ReadFile
[       OK ] BitmapTest.ReadFile (0 ms)
[ RUN      ] BitmapTest.WriteFile
[       OK ] BitmapTest.WriteFile (0 ms)
[ RUN      ] BitmapTest.API
[       OK ] BitmapTest.API (12 ms)
[----------] 3 tests from BitmapTest (13 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test suite ran. (13 ms total)
[  PASSED  ] 3 tests.

Process finished with exit code 0
```

尾声

到此，实验报告的正式内容便终于结束了。尽管我已经尽量精简、挑选重点来讲，但为了读者在理解源代码时更加容易，还是费了不少笔墨。实验过程中最大的感慨便是——当涉及到较为 low-level 的编程时，一切都变得好麻烦啊！尤其是前两个 project 中字节序的转换、buffer 的管理，真是一不小心就会出错，更不用说各种可能情况的讨论了（而且很多情况其实在本实验中并没有被考虑！比如可能跨越多个块的超大记录、主键为变长字段等等）。其实回头一看，核心只是把B+树用块组织在了内存中而已，许多更高级的 trick 并没有实现（比如前缀压缩），而且也并没有支持并发，却已经够累人的了。通过本实验，也算是对偏底层的编程有了一定认识——至少我明白了，low-level 编程代码量一定不小……

最后再提出一点建议：衷心希望老师能够给已有的代码给出更详细的说明，每个 API 也给出更详细一点的文档，这样可以节省许多许多意义不大的阅读代码、猜测架构的时间。而且，很多地方可以明显看出来不够完善——或者说并没有经过重构，比如要释放一个块，既可以直接 `relref`，也可以 `kBuffer.releaseBuf`，类似的情况还有很多，在一开始时真的非常令人困惑。我曾经看过大名鼎鼎的 CMU 15-445 的作业，那真的是对需要实现的 API 作出了非常详尽的说明，希望可以给老师作出参考，让这门挑战性课程越来越好！