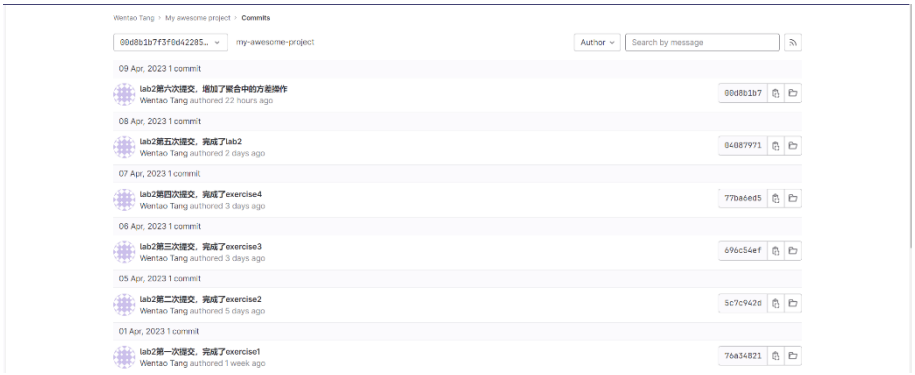
SimpleDB lab2 实验报告

一、Git Commit History



二、设计思路

Exercise1

Predicate 类以及 JoinPredicate 类实现了对 Tuple 中指定字段的运算。
Predicate 类针对 Filter 操作，其构造函数指定了待比较字段在 Tuple 中的序号（field）、比较运算符（op）、参与运算的另一个字段（operand）。该类的核心函数为 filter(Tuple t)，将 t 中第 field 个字段与 operand 进行 op 指定的比较，返回布尔型比较结果。

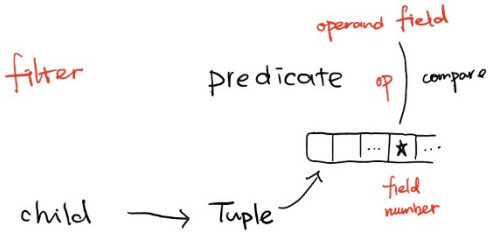


图 1 Predicate

JoinPredicate 类针对 Join 操作，其构造函数指定了两个待比较 field 在 Tuple 中的序号（field1、field2）、比较运算符（op）。该类的核心函数为 filter(Tuple t1, Tuple t2)，将 t1 中第 field1 个字段与 t2 中第 field2 个字段进行 op 指定的比较，返回布尔型比较结果。

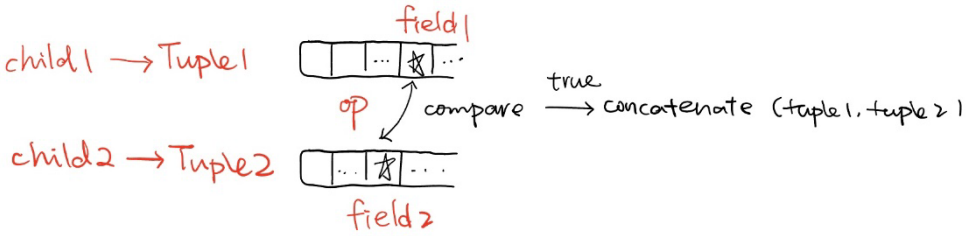


图 2 JoinPredicate

Filter 类实现了对一系列 Tuple 的筛选功能。其构造函数指定了筛选标准 Predicate（p）、输入待筛选 Tuple 的迭代器 child。该类的核心函数为 fetchNext(), 返回下一个符合要求的 Tuple。实现方法为：不断调用 child.next()取出下一个带筛选 Tuple（t），并通过 p.filter(t)判断 t 是否满足条件。

Join 类实现了两表间的 Join 操作。其构造函数指定了筛选标准 JoinPredicate（p）、输入两表中 Tuple 的迭代器 child1 和 child2。该类的核心函数为 fetchNext()，返回下一个符合要求的 Tuple。实现方法为：通过二重循环遍历两表 Tuple 的笛卡尔积，调用 p.filter(Tuple t1,Tuple t2)函数判断当前 Tuple 对（t1,t2）是否满足条件。若满足，则返回拼接后的（t1,t2）。拼接函数为 Tuple 类中的自定义函数 concatenate(Tuple t1,Tuple t2)。

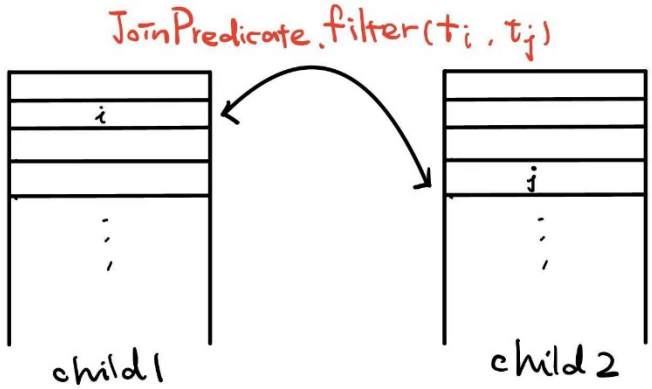


图 3 Join

Exercise2

Aggregate 类实现了对 Tuple 中指定字段的聚合操作，且支持 GROUP BY。其构造函数指定了读入 Tuple 的迭代器 child，分组字段 gfield，聚合字段 afield，聚合操作符 aop。aop 可能是 IntegerAggregator 或 StringAggregator，此类型由 child 的 TupleDesc 决定。该类的核心函数为 fetchnext(), 返回下一个聚合结果。实现时，真正的聚合操作在 open()函数中执行，一次性完成 child 中所有 Tuple 的聚合，并储存在 aop 中，fetchnext 时只需要简单地返回下一个结果即可。

IntegerAggregator 类实现整数类型字段的聚合，聚合结果储存在哈希表 HashMap<Field,Integer> field2int 中，其中键 Field 代表 group by Field，当不需要 GROUP BY 时，会创建一个辅助 Field 保证功能正常。该类的核心函数为 mergeTupleIntoGroup(Tulple tup)，根据构造函数中指定的运算符 op 更新聚合结果。在实现时，利用当前 Tuple 和之前的聚合结果更新，效率较高。

IntegerAggregator 类实现字符串类型字段的聚合，聚合结果同样储存在 HashMap<Field,Integer> field2int 中。该类的 mergeTupleIntoGroup 函数只需要实现 COUNT 操作。

Exercise3

Exercise3 为实现数据库中 Tuple 插入与删除打下了基础，分为 HeapPage、HeapFile、BufferPool 三个层次。
HeapPage 中的 insertTuple 函数向有空位的 Page 中插入 Tuple，实现了 header 的修改、Tuple RecordId 的修改、向 tuples 数组中空位加入 Tuple。对 header 的修改通过 markSlotUsed 函数，对 Byte 数组元素进行位运算实现。
HeapPage 中的 deleteTuple 函数实行懒惰删除策略，只需要找到对应 Tuple 的位置并在 header 中标记该位置为空即可。

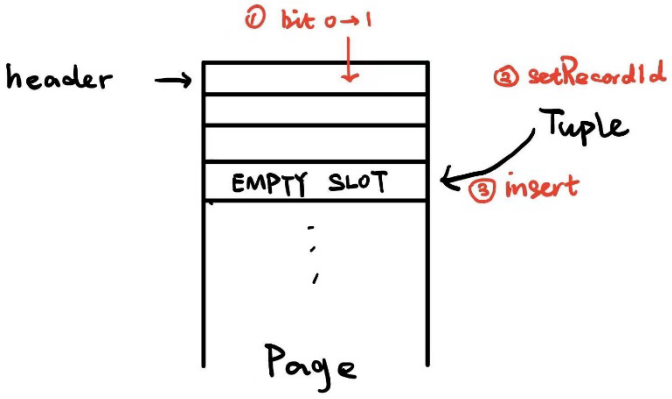


图 4 HeapPage 中的 insertTuple

HeapFile 中的 insertTuple 函数向 BufferPool 请求未满的 Page 并插入 Tuple，返回修改过的 Page。若有未满的 Page（即为 currPage），则向其中插入 Tuple。因为此时并未修改文件，而是返回修改过的 Page，故 currPage 与磁盘中并不相同，标记 currPage 为 dirty。最后将修改后的 currPage 加入到待返回的数组中。
若请求不到未满的 Page，则创建一个新的 Page，并调用 writePage 函数将其写入到该文件末尾。之后通过 BufferPool 的 getPage 获取该页，对该页执行插入、markDirty、加入返回数组操作。
HeapFile 中的 deleteTuple 函数向 BufferPool 请求相应 Page，并调用该 Page 的 deleteTuple 函数，标记该 Page 为 dirty 并将其加入返回数组。

BufferPool 中的 insertTuple 函数将 Tuple 插入指定表，并将修改过的 Page 加入 BufferPool。首先通过 Catalog.getDatabaseFile 函数得到对应文件，并调用该文件的 insertTuple 函数，最后将其返回的 Page 加入到 BufferPool 中。
BufferPool 中的 deleteTuple 函数同样在得到对应文件后，调用该文件的 deleteTuple 函数，将其返回的 Page 加入到 BufferPool 中。

Exercise4

Insert 和 Delete 类实现了对 Tuple 的批量插入、删除操作。
以 Insert 为例，其构造函数指定了输入待插入的迭代器 child，待插入表的 ID tableId。该类的核心函数为 fetchNext()，一次性插入 child 给出的所有 Tuple，并返回单字段 Tuple，指出插入数量。由于逻辑上该方法一次 open 只能插入一轮，故需要设置 hasInserted 标志位，记录本次 open 是否已经插入。delete 类实现逻辑与此类似。

Exercise5

Exercise5 实现了 BufferPool 的 PageEviction 策略，即当 BufferPool 中储存页数量达到上限时，要从磁盘中加入新的 Page 需要先丢弃掉一部分 Page。若将要丢掉的 Page 在 BufferPool 中发生过修改，与磁盘中的副本不一致，即该页为 Dirty，则需要执行 FlushPage，将该页的信息写入磁盘。
flushPage 函数具体实现为取消该页的 dirty 标记，并将该 Page 的信息写入文件对应位置（可通过 PageId 获得），然后更新该 Page 在 BufferPool 中的记录
驱逐策略上，本实验实现了 LRU、MRU 以及 random 策略。LRU 为驱逐最近最少使用的 Page，具体实现为用 HashMap<PageId, Date> id2date 记录 BufferPool 中所有 Page 的最近访问时间，在驱逐时选择时间最早的 Page。该策略适合数据项最近访问得越多，越有可能在接下来出现的场景。
MRU 为驱逐最近最多使用的 Page，实现方法与 LRU 类似。某些情况下，最近越少访问的数据项越有可能在接下来出现，这时 MRU 较合适。
有的场景里，各数据项的出现概念基本相同，均匀地出现，这时 random 策略比较合适。具体实现为生成一个小于 BufferPool 中 Page 总数的随机数 r，并取出遍历 HashMap 时的第 r 项作为驱逐页。
具体策略的选择可通过 BufferPool 类中的 setPolicy 函数实现。

三、改动处
在 IntegerAggregator 中增加了方差操作。由于方差运算较复杂，记录下所有字段每次重新运算开销过大，考虑记录一系列中间变量，利用公式 D(x)=EX^2-(EX)^2 计算。
本实验利用哈希表 avgCount 记录每组的元素个数，avgSum 记录每组元素和，varAvg 记录每组的期望，varRec 记录每组元素的平方和。中间变量都可以实现迭代更新，并且足以计算最终的方差。

四、重难点
1、Operator 中的 hasNext()和 Next()方法实现思路值得借鉴。通过类成员 next 记录下一个元素，如果 next 为 null，则在 hasNext()或 Next()中调用 fetchNext()；否则直接返回 True 或 next 值。该方法只需要消耗一个临时变量的空间，每次调用 hasNext()仅首次需要较多的判断，具有较高的效率。

2、Join 类中的 rewind 函数实现时，除了调用 child1、child2 的 rewind，还必须将辅助变量 child1_curr_tuple 初始化，以完全实现复位的效果。

3、在实现 IntegerAggregator 时，不能用当前平均值更新下一个平均值，因为整数除法有舍入误差，这样每更新一次误差就多一点。所以需要记录组内元素 sum 和元素个数，每次重新计算 avg。

4、在实现 HeapPage 的 insertTuple 函数时，判断待插入 tuple 的 TuldeDesc 与当前 Page 的 td 是否一致必须用 equals。