LAB1 report

521120910245 严冬

1 TupleDesc

1.1 已完成的代码片段解释

- ：实现了返回 `tdItems` 数组的迭代器，允许遍历元组描述中的所有字段项。
- ：该构造函数用于创建一个新的 `TupleDesc` 实例，其中包含了类型数组（`typeAr`）和字段名数组（`fieldAr`）指定的字段。它通过遍历 `typeAr` 数组，并使用每个类型和相应的字段名实例化 `tdItems` 数组中的 `TDItem` 对象。
- ：此构造函数创建一个没有字段名的 `TupleDesc` 实例。它通过遍历 `typeAr` 数组，并为每个类型分配一个空字符串作为字段名来实例化 `tdItems` 数组中的 `TDItem` 对象。
- ：该方法通过字段名来查找字段的索引。如果字段名在 `tdItems` 数组中存在，则返回相应的索引；如果不存在，则抛出 `NoSuchElementException` 异常。
- ：此方法计算根据元组描述中所有字段的类型确定的元组的总字节大小。它通过累加 `tdItems` 数组中每个字段类型的长度来计算大小。
- ：该静态方法将两个 `TupleDesc` 对象合并成一个新的 `TupleDesc` 对象。它首先创建了新的类型和字段名数组，然后将 `td1` 的字段复制到新数组的前部，`td2` 的字段复制到新数组的后部。
- ：该方法用于判断另一个对象是否与当前的 `TupleDesc` 实例相等。它首先检查两个 `TupleDesc` 是否具有相同数量的字段，然后逐一比较它们的字段类型是否相同。
- ：提供了 `TupleDesc` 实例的字符串表示形式，格式为 "fieldName0, ..., fieldNameM"，方便查看和调试。

1.2 整体 TupleDesc 类的作用

`TupleDesc` 类代表了数据库中元组的模式，它定义了元组中每个字段的类型和可选的字段名。这对于数据库操作至关重要，因为它定义了表中数据的结构，允许数据库系统正确解析和存储数据。此类是基础设施的一部分，对于执行查询、存储数据和执行数据类型检查等操作至关重要。

2 Tuple

2.1 已完成的代码片段解释

- ：根据提供的 `TupleDesc` 对象（即元组模式）创建一个新的 `Tuple` 实例。它初始化了字段数组 `fields`，数组的大小由 `TupleDesc` 中的字段数量决定。
- ：返回此元组的 `TupleDesc` 对象，即元组的模式。
- ：返回一个 `RecordId` 对象，表示此元组在磁盘上的位置。可能为 `null`，表示元组可能未存储在磁盘上。
- ：设置此元组的 `RecordId`，更新元组在磁盘上的位置信息。
- ：改变此元组中第 `i` 个字段的值。这里必须保证索引 `i` 是有效的。
- ：返回第 `i` 个字段的值。如果该字段尚未设置，则返回 `null`。
- ：返回此元组的内容的字符串表示形式。格式是每个字段的字符串表示形式之间由空格分隔，最后一个字段后跟一个换行符。
- ：返回一个迭代器，用于遍历此元组的所有字段。
- ：重置此元组的 `TupleDesc`（元组模式），但不会影响到元组中字段的值。

2.2 整体 Tuple 类的作用

`Tuple` 类表示数据库中的单个元组，它根据 `TupleDesc` 对象定义的模式，持有各个字段的数据。这个类是数据库操作的基础，提供了以下关键功能：

- 存储元组的模式和数据。
- 允许读取和修改元组中的每个字段。
- 能够记录和更新元组在磁盘上的位置（通过 `RecordId`）。
- 提供了将元组内容输出为字符串的方法，这对于调试和显示结果很有用。

3 Catalog

3.1 已完成的代码片段解释

- ：初始化了一个新的、空的目录，使用 `ConcurrentHashMap` 来存储表的信息，确保线程安全。
- ：向目录中添加一个新表。如果添加时表名已存在，则移除旧表并添加新表。这样处理名称冲突，确保使用最后添加的表。
- ：根据表名返回表的ID。如果表不存在，则抛出 `NoSuchElementException` 异常。
- ：返回指定表的元组描述符（模式）。如果表不存在，则抛出 `NoSuchElementException` 异常。
- ：返回可以用来读取指定表内容的 `DbFile`。如果表不存在，则抛出 `NoSuchElementException` 异常。
- ：返回指定表的主键字段名。如果表不存在，则抛出 `NoSuchElementException` 异常。
- ：返回一个迭代器，可以遍历目录中表的ID。
- ：根据表ID返回表的名字。如果表不存在，则抛出 `NoSuchElementException` 异常。
- ：从目录中删除所有表。
- ：从文件读取模式并在数据库中创建相应的表。这个方法解析一个文本文件来构建目录，每行代表一个表的模式。

3.2 整体 Catalog 类的作用

`Catalog` 类作为数据库的一部分，负责维护所有表的信息，包括它们的文件引用、表名和主键字段名。它提供了一系列方法来添加、查询和修改表的信息。该类的设计考虑到了线程安全，使用 `ConcurrentHashMap` 以支持可能的并发修改操作。

`loadSchema` 方法是 `Catalog` 的一个特别重要的功能，它允许从一个文本文件中批量加载表模式和数据，这对于初始化数据库和测试非常有用。这个方法解析每个表的字段名和类型，并且为每个表创建了一个 `HeapFile`，这是一个简单的文件格式，用于存储表数据。

`Catalog` 类是数据库的核心组件之一，它使得数据库能够知道它管理的表及其结构。未来，这个类可能会被扩展以从磁盘上的目录表中读取信息，而不是依赖于程序来手动添加表的信息。

4 Bufferpool

4.1 已完成的代码片段解释

- ：创建一个最多可以缓存 `numPages` 页的 `BufferPool`。它通过 `ConcurrentHashMap` 来存储页，确保线程安全。
- ：检索具有相关权限的指定页。如果页在缓冲池中不存在，会从磁盘文件中读取页并将其添加到缓冲池中。如果缓冲池已满，需要执行页替换算法将新页加入缓冲池。

4.2 整体 BufferPool 类的作用

`BufferPool` 类的主要职责包括：

- 管理缓存中页的读取和写入。
- 确保事务在读取或写入页时持有正确的锁。
- 在缓存中查找请求的页，如果找不到，从磁盘中加载。
- 实现页替换策略来处理缓存溢出的情况。

5 heappageid

5.1 已完成的代码片段解释

- ：根据给定的表ID和页码创建一个新的 `HeapPageId` 实例。
- ：返回与此 `PageId` 关联的表的ID。
- ：返回与此 `PageId` 关联的在表中的页码。
- ：返回此页的哈希码，由表ID和页码的字符串连接得到。这对于将 `PageId` 作为哈希表中的键（例如在 `BufferPool` 中）是必要的。
- ：比较一个 `PageId` 与另一个对象是否相等。如果是 `PageId` 并且表ID和页码都相同，则返回 `true`。
- ：返回一个整数数组，表示此对象，用于写入磁盘。返回的数组大小必须包含与构造函数中参数数量对应的整数数量。

5.2 整体 HeapPageId 类的作用

`HeapPageId` 类是数据库系统中的一个基础组件，它提供了以下关键功能：

- 为每个页提供一个全局唯一的标识符，通过表ID和页码组合得到。
- 支持哈希表操作，允许 `HeapPageId` 可以高效地作为键值使用。
- 支持页的等价性比较，即可以判断两个页标识符是否指向数据库中的同一个页。

6 recordid

6.1 已完成的代码片段解释

- ：创建一个新的 `RecordId`，引用指定的 `PageId` 和元组编号。
- ：返回此 `RecordId` 引用的元组编号。
- ：返回此 `RecordId` 引用的页面标识符。
- ：判断两个 `RecordId` 对象是否相等，即它们是否表示相同的元组。相等的条件是它们的 `PageId` 和元组编号相同。
- ：实现 `hashCode()` 方法，以便相等的 `RecordId` 实例（根据 `equals()` 方法）有相同的 `hashCode()`。`hashCode()` 是根据 `PageId` 和元组编号的组合生成的。

6.2 整体 RecordId 类的作用

`RecordId` 类为数据库系统中的元组提供了一个唯一的标识符，它包含以下关键功能：

- 唯一地标识数据库表中的一行数据（即元组）。
- 支持元组的快速查找，因为它包含了元组所在页面的信息以及在该页面中的位置。
- 支持在使用哈希表存储或查找元组标识符时的高效操作。

7 heappage

7.1 已完成的代码片段解释

- ：根据从磁盘读取的数据字节数组创建一个 `HeapPage`。这个构造函数解析页面头部的字节，

用于标识页面上哪些槽（slots）正在使用，然后读取页面上的元组（tuples）。

- ：计算页面上可以有多少个元组。
- ：计算页面头部的字节数。
- ：获取页面修改前的一个视图，用于恢复。
- ：设置当前页面的一个副本作为修改前的视图。
- ：返回与此页面关联的 `PageId`。
- ：从文件中读取下一个元组。
- ：生成表示此页面内容的字节数组，用于将此页面序列化到磁盘。
- ：生成一个对应于空 `HeapPage` 的字节数组，用于向文件中添加新的、空的页面。
- ：从页面中删除指定的元组。
- ：向页面添加指定的元组。
- ：标记此页面为脏页面或非脏页面，并记录造成脏页面的事务。
- ：返回最后使此页面变脏的事务的ID，如果页面不是脏的，则返回 `null`。
- ：返回此页面上空槽的数量。
- ：返回指定槽是否正在使用。
- ：标记页面上的一个槽为已使用或未使用。
- ：返回一个迭代器，遍历此页面上的所有元组（注意，这个迭代器不应该返回空槽中的元组）。

7.2 整体 HeapPage 类的作用

`HeapPage` 类是 SimpleDB 数据库系统中的一个核心组件，它提供了以下关键功能：

- 管理页面上的元组，包括元组的添加和删除。
- 维护页面头部的元数据，以标识哪些槽正在使用。
- 提供序列化页面到磁盘和从磁盘反序列化页面的功能。
- 支持恢复机制，通过跟踪页面修改前的状态。

8 heapfile

8.1 已完成的代码片段解释

- ：构造函数通过指定的文件 `f` 来后置此 `HeapFile`。
- ：返回磁盘上支持此 `HeapFile` 的 `File` 对象。
- ：返回唯一标识此 `HeapFile` 的ID。建议通过对文件绝对路径名的哈希值来生成这个ID。
- ：返回存储在此 `DbFile` 中的表的 `TupleDesc` 对象。
- ：从磁盘读取指定ID的页面。
- ：将页面写回磁盘。对于实验1（lab1）不是必需的。
- ：返回此 `HeapFile` 中页面的数量。
- ：在 `HeapFile` 中插入一个元组。对于实验1不是必需的。
- ：从 `HeapFile` 中删除一个元组。对于实验1不是必需的。
- ：返回一个 `DbFileIterator`，它可以遍历 `HeapFile` 中的元组。

8.2 `HeapFileIterator` 辅助类

- ：`HeapFile` 的内部类，实现了 `DbFileIterator` 接口，用来迭代 `HeapFile` 中的元组。
- ：打开迭代器，准备开始迭代页面中的元组。
- ：获取指定页面的元组迭代器。
- ：如果还有更多的元组可以迭代，返回 `true`。
- ：返回迭代器中的下一个元组。
- ：重置迭代器，从头开始迭代元组。
- ：关闭迭代器。

8.3 HeapFile 类的作用

`HeapFile` 类提供了以下关键功能：

- 封装文件I/O操作，允许从磁盘读取和写入页面。
- 提供页面的数量和页描述。
- 支持数据库的插入和删除操作，虽然在实验1中这些方法不是必需的。
- 提供一个迭代器，以便能够遍历文件中的所有元组。

9 seqscan

9.1 已完成的代码片段解释

- ：创建一个在指定事务中，对指定表进行顺序扫描的 `SeqScan`。
- ：返回操作器扫描的表的实际名称。
- ：返回此操作器扫描的表的别名。
- ：重置此操作器的 `tableId` 和 `tableAlias`。
- ：打开迭代器准备进行扫描。
- ：返回基础 `HeapFile` 的 `TupleDesc`，字段名称前缀为构造函数中的 `tableAlias` 字符串。
- ：如果还有更多元组可读，则返回 `true`。
- ：返回下一个元组，如果没有更多元组，则抛出 `NoSuchElementException`。
- ：关闭迭代器。
- ：重置迭代器到初始状态，准备重新开始扫描。

9.2 SeqScan 类的作用

`SeqScan` 类提供了以下关键功能：

- 使得可以在一个指定的事务上对一个数据库表进行顺序扫描。
- 支持表的别名处理，这在执行表连接操作时特别有用，因为可以区分具有相同字段名的不同表。
- 提供了数据库表的迭代器接口，使得可以通过迭代器模式来访问表中的元组。