# CS 660 Programming Assignment 1 Report

*Name: Zhiqi Jin*

## TDItem Class

The TDItem class represents a single item in a tuple description. This class is used to store information about a field's type and name in a database table. Here's an overview of its key functionalities:

bool operator==(const TDItem &other) const: This method compares two TDItem objects for equality based on their string representations, which include the field type and field name.

std::size_t std::hash<TDItem>::operator()(const TDItem &r) const: This function defines a custom hash function for TDItem objects, allowing them to be used in hash-based data structures.

## TupleDesc Class

The TupleDesc class represents the description of a tuple in a database table. It includes information about the types and names of fields within the tuple. Here's an overview of its key functionalities:

TupleDesc(const std::vector<Types::Type> &types): This constructor creates a TupleDesc object with field types provided in the input vector. Field names are initialized to empty strings.

TupleDesc(const std::vector<Types::Type> &types, const std::vector<std::string> &names): This constructor creates a TupleDesc object with field types and names provided in separate input vectors.

size_t numFields() const: This method returns the number of fields in the tuple description.

std::string getFieldName(size_t i) const: Given an index i, this method retrieves the name of the field at that index.

Types::Type getFieldType(size_t i) const: Given an index i, this method retrieves the type of the field at that index.

int fieldNameToIndex(const std::string &fieldName) const: Given a field name, this method returns the index of the corresponding field in the tuple description.

size_t getSize() const: This method calculates and returns the total size in bytes of the tuple based on the sizes of its individual fields.

TupleDesc merge(const TupleDesc &td1, const TupleDesc &td2): This static method merges two TupleDesc objects, td1 and td2, to create a new TupleDesc that represents the combined fields of both.

std::string to_string() const: This method generates a string representation of the TupleDesc object, including all its field descriptions.

bool operator==(const TupleDesc &other) const: This method compares two TupleDesc objects for equality based on the number of fields and their types.

iterator begin() const: This method returns an iterator pointing to the beginning of the TupleDesc's field items.

iterator end() const: This method returns an iterator pointing to the end of the TupleDesc's field items.

std::size_t std::hash<db::TupleDesc>::operator()(const db::TupleDesc &td) const: This function defines a custom hash function for TupleDesc objects, allowing them to be used in hash-based data structures.

## Tuple Class

The Tuple class represents a tuple in a database table. A tuple is essentially a collection of fields, each of which has a specific data type. The class encapsulates the following functionalities:

Tuple(const TupleDesc &td, RecordId *rid): This constructor initializes a Tuple object with a given TupleDesc and a pointer to a RecordId. It ensures that the TupleDesc has at least one field, and it initializes the contents of the tuple with null pointers, one for each field.

const TupleDesc &getTupleDesc() const: This method returns a reference to the TupleDesc associated with the tuple, which describes the data types and names of the fields.

const RecordId *getRecordId() const: This method returns a pointer to the RecordId associated with the tuple, which identifies the tuple's location in the database.

void setRecordId(const RecordId *id): This method allows you to set the RecordId for the tuple, typically used when the tuple is inserted into the database.

const Field &getField(int i) const: Given an index i, this method retrieves the field at the specified index in the tuple. It returns a reference to the field.

void setField(int i, const Field *f): Given an index i and a pointer to a Field, this method sets the field at the specified index in the tuple. It also performs boundary checks to ensure that the index is valid.

iterator begin() const: This method returns an iterator pointing to the beginning of the tuple's field contents.

iterator end() const: This method returns an iterator pointing to the end of the tuple's field contents.

std::string to_string() const: This method generates a string representation of the Tuple object. It iterates through the tuple's fields, converting each field to its string representation and concatenating them.

## Catalog Class

The Catalog class manages information about tables in a database, such as table names, primary keys, and associated database files. It provides methods to add, retrieve, and clear table information. Here's an overview of its key functionalities:

void addTable(DbFile *file, const std::string &name, const std::string &pkeyField): This method adds a new table to the catalog. It takes a pointer to a DbFile, the table's name, and the primary key field name as input. If a table with the same name or the same file ID exists in the catalog, it is replaced.

int getTableId(const std::string &name) const: Given a table name, this method retrieves the table's ID (file ID). If the table name is not found in the catalog, it throws an exception.

const TupleDesc &getTupleDesc(int tableid) const: Given a table ID (file ID), this method retrieves the TupleDesc associated with the specified table. If the table ID is invalid, it throws an exception.

DbFile *getDatabaseFile(int tableid) const: Given a table ID (file ID), this method retrieves the corresponding DbFile associated with the specified table. If the table ID is invalid, it throws an exception.

std::string getPrimaryKey(int tableid) const: Given a table ID (file ID), this method retrieves the primary key field name associated with the specified table. If the table ID is invalid, it throws an exception.

std::string getTableName(int id) const: Given a table ID (file ID), this method retrieves the name of the table associated with the specified ID. If the table ID is invalid, it throws an exception.

void clear(): This method clears the catalog, removing all stored table information.

## HeapPage Class

The HeapPage class represents a page in a heap file within a database system. It is responsible for managing and storing tuples on the page. Here's an overview of its key functionalities:

HeapPage(const HeapPageId &id, uint8_t *data): This constructor initializes a HeapPage object with a given HeapPageId and raw page data. It reads the header information, initializes the TupleDesc, and reads the tuples from the raw data.

int getNumTuples(): This static method calculates and returns the maximum number of tuples that can fit on a page based on the page size and tuple size.

int getHeaderSize(): This static method calculates and returns the size of the header section of the page.

PageId &getId(): This method returns the HeapPageId associated with the page.

void readTuple(Tuple *t, uint8_t *data, int slotId): This method reads a tuple from raw data and initializes a Tuple object with the appropriate TupleDesc and RecordId. It extracts fields from the raw data and sets them in the Tuple.

void *getPageData(): This method creates and returns a new block of memory containing the entire page's data, including the header and tuples, in a format suitable for storage.

uint8_t *createEmptyPageData(): This static method creates an empty page of data filled with zeros.

int getNumEmptySlots() const: This method counts and returns the number of empty slots (unused tuples) on the page.

bool isSlotUsed(int i) const: This method checks if a slot (tuple) at a given index is used (occupied) on the page.

HeapPageIterator begin() const: This method returns an iterator pointing to the beginning of the page's tuples.

HeapPageIterator end() const: This method returns an iterator pointing to the end of the page's tuples.

## HeapPageIterator Class

The HeapPageIterator class provides an iterator interface to iterate over the tuples on a HeapPage. Here's an overview of its key functionalities:

HeapPageIterator(int i, const HeapPage *page): This constructor initializes a HeapPageIterator object with a starting slot index and a pointer to the associated HeapPage. It automatically advances to the next used slot if the initial slot is empty.

bool operator!=(const HeapPageIterator &other) const: This method checks for inequality between two HeapPageIterator objects based on their slot and page references.

HeapPageIterator &operator++(): This method increments the iterator to the next used slot on the page, skipping empty slots.

Tuple &operator*() const: This method returns a reference to the tuple currently pointed to by the iterator.

## HeapFile Class

The HeapFile class represents a heap file in a database, which is responsible for managing and storing data pages. Here's an overview of its key functionalities:

HeapFile(const char *fname, const TupleDesc &td): This constructor initializes a HeapFile object with a given file name (fname) and tuple description (td). It reads data from the file, calculates the number of pages, and initializes pages for reading.

uint8_t *readData(const char *fname): This private method reads binary data from a file with the given name (fname) and returns it as a pointer to uint8_t. It also calculates and stores the data size.

int generateFid(const char *fname): This private method generates a file ID (fid) by hashing the tuple description and the file name.

int getId() const: This method returns the file ID (fid) associated with the heap file.

const TupleDesc &getTupleDesc() const: This method returns the tuple description (td) associated with the heap file.

Page *readPage(const PageId &pid): This method reads a specific page from the heap file using a given PageId. It copies the data for the page from the heap file's data buffer and returns a HeapPage object.

int getNumPages(): This method calculates and returns the total number of pages in the heap file based on the data size and page size.

const Page *getPage(int index): This method returns a pointer to a specific page within the heap file, identified by its index.

HeapFileIterator begin() const: This method returns an iterator pointing to the beginning of the heap file's pages.

HeapFileIterator end() const: This method returns an iterator pointing to the end of the heap file's pages.

## HeapFile Class

The HeapFile class represents a heap file in a database, which is responsible for managing and storing data pages. Here's an overview of its key functionalities:

HeapFile(const char *fname, const TupleDesc &td): This constructor initializes a HeapFile object with a given file name (fname) and tuple description (td). It reads data from the file, calculates the number of pages, and initializes pages for reading.

uint8_t *readData(const char *fname): This private method reads binary data from a file with the given name (fname) and returns it as a pointer to uint8_t. It also calculates and stores the data size.

int generateFid(const char *fname): This private method generates a file ID (fid) by hashing the tuple description and the file name.

int getId() const: This method returns the file ID (fid) associated with the heap file.

const TupleDesc &getTupleDesc() const: This method returns the tuple description (td) associated with the heap file.

Page *readPage(const PageId &pid): This method reads a specific page from the heap file using a given PageId. It copies the data for the page from the heap file's data buffer and returns a HeapPage object.

int getNumPages(): This method calculates and returns the total number of pages in the heap file based on the data size and page size.

const Page *getPage(int index): This method returns a pointer to a specific page within the heap file, identified by its index.

HeapFileIterator begin() const: This method returns an iterator pointing to the beginning of the heap file's pages.

HeapFileIterator end() const: This method returns an iterator pointing to the end of the heap file's pages.