CSE 510: Database Management System Implementation

Project Phase 2: Report

Submitted by Group 5

Group Members:

Sucharitha Rumesh
Supriyaa Damodaraswamy
Spoorthi Karnati
Naga Subrahmanya Sai Tarun Malladi
Hariharan Adhithya Venkatraman
Thiviyakalyani Navaneethan
Ashish Agarwal

Abstract:

A Database Management System handles a collection of data, their storage, update and access. In this Phase of the project, we aim to implement a Column-Oriented Database leveraging the modules of Minibase for Java. Minibase is a relational Database Management System that is a Row-oriented Database. In Column-store, each column or attribute value is stored contiguously. They perform better on I/O as only the necessary attributes have to be accessed. Modifications to the Disk and Buffer modules are needed. Implementation of additional modules for columnar storage and Indexing like Columnar Database implementation, Columnar file implementation and Bitmap index implementation are done. Batch insert does insertion of a large number of records in a columnar database. Index query creates two types of index – Bitmap and BTree on a given column. Query retrieves matches from the database based on constraint using different access type. Delete eliminates matches from the database based on a constraint.

Keywords: Columnar Database, Columnar file, Bitmap, Batch Insert, Index scan, Tuple scan, Btree, Disk Manager.

Introduction:

Relational Database Management System is traditionally implemented as a Row-store. When querying is done in such system we have too read the whole record and do a projection of the required attributes. This is an expensive operation when there are thousands of records and only very few attributes being projected. It is also not feasible to use this type of database when the schema is frequently changing, with addition and deletion of attributes. This will require accessing each record to delete the attribute or creating space for each record for the extra attribute.

When presented with such a situation, implementation of the Relational Database as a Column-store is an effective solution. In Column-oriented Database, the records are split into columns and stored in separate columnar files. A specific column data can be accessed contiguously when querying. Each value in a columnar file is accessed using its Tuple id by initiating a Tuple scan. Indexing of Column store is done by extending the BTree index of row store. Here the leaf data page stores the columnar file values.Column store is more effectively indexed using a Bitmap. Each column will have a number of Bitmaps depending on the number

of unique values in that column. In Bitmap, a bit is set if the value in the corresponding position in the columnar file is equal to the value on which the bitmap is being created.

In this Phase of the project, we aim to implement four query functions on Columnar Data. The Batch Insert query inserts a number of data records from a data file in the columnar database by creation of columnar files for each column. Index query creates index on the given column in a columnar database. This query takes two types of index as arguments, namely Bitmap and Btree. Btree creates a tree on a given column and Bitmap creates a Bitmap file for each unique value in the given column. Query program fetches matching results in the Target columns from the Columnar database based on value constraint on some column. This query uses different type of access methods like Filescan, Columnscan, Bitmap and Btree. Delete Query Program is similar to the Query program except it eliminates the matching tuples from the database and mentions whether the deleted value will be purged.

Implementation:

Value Class:

An abstract class used to store the value depending upon its data type. As per specification for now we are dealing with Integer and String values. Therefore, the class has been extended further into two classes as 'IntegerValue' class and 'StringValue' class. Value class can be extended further to support other data types also.

IntegerValue Class:

This class is an extension of Value class to handle integer data values. Member variables and member functions of the class are as follows:

Member Variable: int value
Member Functions: getValue( ) function is used to return the integer value stored in the member variable value. setValue( ) sets the value of the member variable value to value of the passed parameter.

StringValue Class:

An extension of Value class to handle string data values. Member variables and member functions of the class are as follows:

Member Variable: String value

Member Functions: getValue( ) function is used to return the String value stored in the member variable value. setValue( ) sets the value of the member variable value to the value of the passed parameter.

TID Class:
This class is the Tuple identifier class. It is analogous to the RID class predefined in the Minibase global package.

Member Variables:
This class contains three variables - position, number of records and array of record ids corresponding to the columns.

Member Functions:

TID( ): Constructor for TID object initialization are overloaded for different number of parameters.

getposition( ): This function returns the current value of position.

setposition( ): The value of member variable position is set to the given parameter value in this function.

copyTID(): Creates a copy of the given tid.

equals( ):

- This is a boolean function which checks for the equality of the two objects.
- First it checks whether the two tuple ids have same number of records and their position is same or not. If yes, then it further checks for the equality of the given record ids page id and slot id if all the records are matched then returns true else it return false.

writeToByteArray( ):

- This is a function to write the TID into the given data buffer.
- For writing, the predefined functions of the minibase - Convert class were used as it provides conversion functions for getting and setting data in byte array at particular position.

ColumnarFile Class:

ColumnarFile class analogous to HeapFile class of the minibase. This class is a part of the Columnar Package.

Member Variables:

- String columnarFileName: name of the columnar file given as input.
- int numberOfColumns: number of columns in the given data file.
- AttrType[] attributeType: data types for the columns in the given data file.
- int stringSize: the size of the string column as parsed from the given data file.
- int tupleLength: the total length of tuple (cumulative size of all the fields in the given data file).
- int deleteCount: counter to store the number of records deleted.
- String [] heapFileNames: the names of the heap files.
- String [] columnNames: the names of the columns.
- Heapfile[] heapFileColumns: heap file to store the column data corresponding to each column.
- Heapfile columnarFile: heapfile that is the actual Columnar file.
- Heapfile deletedTupleList: heapfile to store deleted values.
- ColumnarFileMetadata cfm: columnar file metadata object to store the statistics related to the created columnar file.

Member Functions:

ColumnarFile():

- This is a constructor used to initialize the columnar file object with the given columns name, number of columns and the types of the column provided.
- All the heapfiles corresponding to each column gets created.
- Depending upon the number and type of the columns in the given data tuple length is calculated (4 for integer and 25 for string).
- Also the columnar heap file is created.
- Columnar metadata datafile is also created to store the metadata.

deleteColumnarFile(): This function is used to delete the heapfiles that were created for each column.

insertTuple():

- This function is used to insert tuple data into the heap file using the data stored in the given byte array provided to it.
- Create a new TID object and initialize it.
- A curPos variable is used to keep track of the position in the given byte array.

- Check for the attribute type of columns, if found integer then the pointer moves the current position and fetches the integer value from the byte array using the predefined Convert class functions and position gets incremented by 4 (as it is an integer). A new byte array is created to store the integer value fetched in the previous step. Further a new record id is created to store the value and value in written into the corresponding heap file using the predefined function of the heapfile.
- The same procedure as above is followed for String except that the position is incremented by different value.

getTuple():

- The tuple with the given TID is fetched in this function.
- Open the columnar heap files associated with the given TID and check for the datatype of the attribute.
- Depending on the attribute type, value is fetched from the heapfile and stored into the new tuple object and offset value is updated.Same procedure is followed for the string.
- Tuple fetched is returned at the end.

getValue():

- Fetches the value from the given tuple id and column number.
- First fetch the data from the heap file corresponding to the given heapfile and column number.
- Checks for the type of the column and fetches the value using the Convert class function and store it into the value class object (integer or string).
- Return the value.

getTupleCnt( ): This function opens the given columnar file and counts the number of records in it using the predefined heapfile functions.

openTupleScan( ): This function is used to initiate sequential scan of the tuples using the TupleScan object.

openColumnScan( ): Initialization of sequential scan along the given columns number is done in this method.

getHeapfileForColumname( ): Returns the heap file for the given column name.

getColumnNumberFromColumname(): Returns the column number for the given column name.

setColumnarFileMetadata(): This function sets the metadata of the current columnar database in the columnar metadata object.

getColumnarFileMetadata(): Fetching the columnar file metadata for the given column name is done in this function.

markTupleDeleted(): This function keeps track of all the deleted tuples and adds them to a heapfile.

purgeAllDeletedTuples(): Actual delete of all the marked tuples is done in this function.

TupleScan Class:
This Class is used to scan the tuples.It is analogous to the Scan class. This class is a part of the Columnar Package.

Member Variables:
A columnar file object and scan variable.

Member Function:

TupleScan( ):

- This is a Default constructor which takes columnar file as its argument.
- It Initiates the scan along all the columns of the file.

closetuplescan( ): Closes all the scans that were initiated for the current tuplescan object.

getNext( ):

- This function fetches the next tuple form the heap file using the TID.
- It looks into all the column heapfiles associated with the given TID and based on the data type fetches the value using the Convert class and stores it into a tuple.
- The tuple containing the values from all the columns is returned.

position( ):

- Function is used to position the pointer on the given position in all the column files.

ColumnarFileMetadata Class:
A metadata class that stores all the metadata corresponding to the given database. This class is a part of the Columnar Package.

Member Variables:

- String columnarFileName: name of the columnar file given as input.
- int numberOfColumns: number of columns in a given data file.
- AttrType[] attributeType: data types for the columns in the given data file.
- int stringSize: the size of the string column as parsed from the given data file.
- int tupleLength: the total length of tuple (cumulative size of all the fields in the given data file).
- int deleteCount: counter to store the number of records deleted.
- String [] heapFileNames: the names of the heap files.
- String [] columnNames: the names of the columns.
- Heapfile[] heapFileColumns: heap file to store the column data corresponding to each column.
- byte[] data: byte array used to store the data.

Member Functions:

ColumnarFileMetadata( ): Initializes metadata object using the given columnar file object.

ColumnarFileMetadata ( ):

- Initialize the metadata object using the given column name.
- Opens the column heap file and scans one record from it .
- Calls the getColumnarFileMetadata( ) function to get the metadata.

getColumnarFileMetadata( ): This function fetches the data using the tuple variable.

getTuple(): This function parses through the byte array and fetches the data from it.

BM Class:
This Class is analogous to BT class in BTree Package. This class is a part of the bitmap Package.

Member Functions:

pinPage( ):

- This method with Page as the return type, is used to pin the page that is currently being accessed using the page ID and its page number.
- The corresponding pinpage method from DB class is accessed by using the SystemDefs and JavabaseBM (Buffer Manager Object) from global package.
- If the page is not pinned, it is returned.
- PinPageException is used and printStackTrace() method is inherited.

unPinPage( ):

- This method with Page as the return type, is used to unpin the page i.e. the page is not dirty (no current updates have been made).
- The corresponding unpinpage method is accessed from DB class by using the SystemDefs and JavabaseBM (Buffer Manager Object) from global package.
- UnPinPageException is used and printStackTrace() method is inherited.

printBitMap( ) :

- This function is used for printing the bitmap in the BitMapFile that has the given header as its Header Page.
- It gets the rootId( ) from the header and check if it is a Valid page.
- If the pid at root is INVALID_PAGE, then it prints that the bitmap is empty.
- Otherwise it calls the _printPage( ) function with the root Id as the parameter.

_printPage( ):

- This function pins the page at the PageId that it gets as input.
- It accesses the data at this BMPage, using the getBMpageArray( ) function in BMPage.
- It prints the Bitmap which is stored from the StartByte to StartByte + Count in the byte data Array.
- The value of StartByte is retrieved using the getStartByte( ) function of BMPage and Count is retrieved using the getCount( ) function of BMPage.
- It gets the PageId of the next page by using the getNextPage( ) function and stores it in apage.
- It unpins the page that was pinned initially.
- It calls _printPage( ) with apage as the parameter.

getpositions( ):

- This function is used for printing the bitmap in the BitMapFile that has the given header as its Header Page.
- It gets the rootId( ) from the header and check if it is a Valid page.
- If the pid at root is INVALID_PAGE, then it prints that the bitmap is empty.
- Otherwise it calls the getpos( ) function with the root Id as the parameter.

getpos( ):

- This function pins the page at the PageId that it gets as input.
- It accesses the data at this BMPage, using the getBMpageArray( ) function in BMPage.
- It traverses the Bitmap which is stored from the StartByte to StartByte + Count in the byte data Array and stores the positions where bit is set in the bitmap.
- The value of StartByte is retrieved using the getStartByte( ) function of BMPage and Count is retrieved using the getCount( ) function of BMPage.
- It gets the PageId of the next page by using the getNextPage( ) function and stores it in apage.
- It unpins the page that was pinned initially.
- It calls getpos( ) with apage as the parameter.

getCount( ):

- Returns the value of c.

BitMapHeaderPage

This Class is analogous to BTreeHeaderPage in BTree Package. This class is a part of bitmap Package. It extends HFPage.

Member Variables:

int len: a static variable used to get the length of the string value.

Member Functions:

setPageId( ): This function calls the setCurPage( ) with PageId which it takes as input.

getPageId( ): This function returns the value it gets from calling getCurPage( ).

set_magic0( ): This function calls setPrevPage( ) with a new PageId set to magic which it gets as input.

get_magic0( ): This function gets the PageId at Previous position using getPrevPage( ) and returns its pid .

set_rootId( ): This function calls setNextPage( ) with rootId which we get as an input.

get_rootId( ): The function getNextPage( ) is called to retieve the PageId of the root page.

set_ColNo( ): This function calls the setSlot( ) function and essentially stores the given colNo at slot 1 at offset 0.

get_ColNo( ): This function calls the getSlotLength( ) to get the value of ColNo which is stored in slot 1.

set_value( ): This function calls the setSlotvalue( ) function and stores the given parameter value at slot 2 at offset 0. It computes len as the length of the argument value + 2.

get_integervalue( ): The function get_intval( ) is called with argument 2 to get the value at slot 2.

get_stringvalue( ): The function get_stval( ) is called with argument 2 and len to get the value at slot 2.

BitMapHeaderPage( ):

- The constructor with PageId as its parameter.
- The pinpage method from BufferManager class is accessed by using the SystemDefs and JavabaseBM (Buffer Manager Object) from global package to pin the page with given PageId.

BitMapHeaderPage( ): The constructor with page as its parameter, calls super( ) of page.

BitMapHeaderPage( ):

- calls super( ). Gets a new Page variable.
- Opens a new page using newpage method from Buffer Manager class is accessed by using the SystemDefs and JavabaseBM (Buffer Manager Object) from global package and returned PageId is stored.
- Calls init with PageId and apage.

BitMapFile class :

The BitMapFile Class creates a BitMapFile for each unique value in a column. It is analogous to the BtreeFile class. It is created under the Bitmap package.

Member Variables:

- int MAGIC0: used for sanity checking and detecting corruptions during updates.
- FileOutputStream fos: used to create an output stream in order to write data into the BitMapFile.
- DataOutputStream trace: used to write the data into the BitMapFile.
- BitMapHeaderPage headerPage: stores the Header Page of the BitMapFile.
- PageId headerPageId: store the page ID of the header page.
- String FileName: contains name of the BitMapFile that is currently used.

Member Functions:

traceFilename( ):

- This is a function which helps to traces the file using the filename provided as a parameter and created a FileOutputStream and a DataOutputStream to enable writing data into the file.
- The output is used to show the inner workings of the Bitmap during its operations.

destroyTrace( ): This function stops the trace of a file and closes the file.

getHeaderPage(): with BitMapHeaderPage as the return type, this is an access method used access the headerPage of the BitMapFile.

get_file_entry( ):

- This method has PageId as the return type and gets fileEntryname which is the BitMapFile name as the parameter.
- This will access the get_file_entry in the DB class by using the SystemDefs and JavabaseDB (DB Object).
- This method will return the header page ID of the given BitMapFile.

add_file_entry( ):

- With PageId as the return type, this method will take in the BitMapFile name and the header page ID of the BitMapFile.
- This method adds a file entry for the newly created BitMapFile by accessing the add_file_entry( ) in the DB class by using the SystemDefs and JavabaseDB (DB Object).

pinPage( ):

- This method with Page as the return type, is used to pin the page that is currently being accessed using the page ID and its page number.
- The corresponding pinpage method from DB class is accessed by using the SystemDefs and JavabaseBM (Buffer Manager Object) from global package.
- If the page is not pinned, it is returned.
- PinPageException is used and printStackTrace() method is inherited.

unPinPage( ):

- This method with Page as the return type, is used to unpin the page i.e. the page is not dirty (no current updates have been made).
- The corresponding unpinpage method is accessed from DB class by using the SystemDefs and JavabaseBM (Buffer Manager Object) from global package.
- UnPinPageException is used and printStackTrace() method is inherited.

freePage( ):

- This method is used to free the Bit Map Page.
- This corresponding freepage method is accessed from the BuffMgr class by using SystemDefs and JavabaseBM (Buffer Manager Object) from global.
- FreePageException is used by inheriting the printStackTrace() method.

delete_file_entry( ):

- This method is used to delete the file from the database.
- The corresponding delete_file_entry() method is accessed from the DB class by using the SystemDefs and JavabaseDB (DB Object) from the global package.
- DeleteFileEntryException is used and printStackTrace() method is inherited.

BitMapFile( ):

- String file_entry_name is passed as a parameter which is the name of the existing BitMapFile we want to access.
- The headerPageId created initially is used to store the Page ID that is returned by the get_file_entry( ) function by providing the BitMapFile name.
- BitMapHeaderPage constructor is used to access the Headerpage referred by the headerPageId retrieved from the previous step.
- Now, FileName object will contain the BitMapFile's name.
- GetFileEntryException, ConstructPageException, IOException, AddFileEntryException, InvalidTupleSizeExceptionand FieldNumberoutOfBoundException are used.

BitMapFile(String file_entry_name, ColumnarFile columnfile, int column, ValueClass value) :

- This Constructor is used to create a new BitMapFile.
- BitMapFile's name, Columnarfile's name, the column number and the value on which the BitMap has to be created are passed as the parameters.
- It checks if the BitMapFile exists or not by checking if the header page ID of the Bit map is null or not.
- If the header page is NULL, then a new BitMapHeaderPage is created and its ID is retrieved by using the getPageId( ) function and stored in headerpageId object.
- Using the BitMapFile name and its newly created header page ID, add_file_entry( ) is called.
- Using the headerPage object, set_magic0 function is accessed to set the magic number to MAGIC0.
- set_rootId is accessed which will return the Page ID of the new page and setting it to an INVALID_PAGE.
- set_value function is accessed to set the value for which the bitmap has to be created.
- If the header page is not null, then we are accessing the Bit map header page by passing the headerPageId as the parameter.
- The BitMapFile's name is stored in the FileName.
- If the value for which the bitmap is created is of the type IntegerValue, then it is stored in the integer variable intkeyval by using the .getValue() function. Using the column file name, column number and intkeyval, accessInt function is called.
- Similarly, if the value is of the type String, the value is stored in strkeyval (String) and accessStr function is called by using the column file name, column number and strkeyval.
- GetFileEntryException, ConstructPageException, IOException, AddFileEntryException, InvalidTupleSizeException, FieldNumberOutOfBoundException are the exception used.

accessStr( ):

- A new Tuple object t, integer variable position, RID object rid is created.
- Columnarfile object is used to access the openColumnScan by passing the columno as the parameter. The scan that is returned is stored in the columnScan, which is a Scan Object.
- A while loop is used which will scan through each value to retrieve the next record in a sequential scan.
- setTupleMetaData is called by using t, to set the tuple length, number of columns and the field Offset.
- ColVal, a String variable is used to store the string value from the scan.
- If the ColVal and the value passed as a parameter are the same, insert function is called. Otherwise delete function is called.
- Position variable is incremented and the while loop runs again till the value returned on scan is null.
- InvalidTupleSizeException, IOException, FieldNumberOutOfBoundException are used.

accessInt( ):

- A new Tuple object t, integer variable position, RID object rid is created.
- Columnarfile object is used to access the openColumnScan by passing the columno as the parameter. The scan that is returned is stored in the columnScan, which is a Scan Object.
- A while loop is used which will scan through each value to retrieve the next record in a sequential scan.
- setTupleMetaData is called by using t, to set the tuple length, number of columns and the field Offset.
- ColVal, an Integer variable is used to store the integer value from the scan.
- If the ColVal and the value passed as a parameter are the same, insert function is called.
- If not, delete function is called.
- Position variable is incremented, and the while loop runs again till the value returned on scan is null.
- InvalidTupleSizeException, IOException, FieldNumberOutOfBoundException are used.

close( ):

- If the header page is not null, i.e. there exist a BitMapFile, then by using the SystemDefs and JavabaseBM (Buffer Manager Object) is used to access the unpinPage function to unpin the page and close the file.
- Headerpage is then set to null.

- PageUnpinnedException,InvalidFrameNumberException, HashEntryNotFoundException and ReplacerExceptions are used.

insert( ):

- This method is used to insert bit 1, in the given position inside the bitmap. Its return type is Boolean. It returns true if it was successfully inserted.
- apage, a new PageId object is created.
- There are two cases for which this insert function works, (1) when the root ID of the header page is INVALID and (2) when the root ID of the header page is VALID.
- When the root ID is invalid, a new BMPage is created (page).
- Using page, setNextPage() function is accessed in order the set the next page as INVALID.
- Page is again used to getCurpage() i.e. to get the current page ID and that is set as the root ID of the Bitmap header pager.
- setBit(position,1) function from the BMPage class is called to insert the bit 1, in the position mentioned.
- Now, when the root ID is VALID, then p, a new PageId object is created and the headerpage ID is stored in it.
- Using the p, we are pinning the page to pg1, a Page variable.
- Using the pinned page, the already existing BMPage is accessed and its next page is set to INVALID.
- Now, available_space() function is used to check if the data is already present in the page or not.
- If data is already present, and still there is some available space in the page, then the number of values inserted in the page, which is the count is stored in the integer variable count.
- If the position entered is greater than the number of data present in the page, then the position given is invalid and insert won't be performed.
- If not, setbit(position,1) is called and bit 1 is set in the given position successfully.
- If the data is present and there is no available space in the page, then a new BMPage (page1) and PageId (apage1) is created.
- The previous page, i.e. page, is used to set its next page as the new page which is created (page1) and page1 is used to set its previous page as page. Apage1 is set as page1's page ID.
- By using apage1, a new Page variable, p2 is created so as to pin the page using apage1.
- Using p2, the already existing BMPage is accessed, and the its next page is set as INVALID.

- If there is space available, then just like the previous process using count, insertion is successfully implemented.
- After insertion is done, the page that is been currently pinned (modified) is unpinned.

delete( ):

- This method is used to insert bit 0, in the given position inside the bitmap. Its return type is Boolean.It returns true if it was successfully deleted.
- apage, a new PageId object is created.
- There are two cases for which this insert function works, (1) when the root ID of the header page is INVALID and (2) when the root ID of the header page is VALID.
- When the root ID is invalid, a new BMPage is created (page).
- Using page, setNextPage() function is accessed in order the set the next page as INVALID.
- Page is again used to getCurpage() i.e. to get the current page ID and that is set as the root ID of the Bitmap header pager.
- setBit(position,0) function from the BMPage class is called to insert the bit 0, in the position mentioned.
- Now, when the root ID is VALID, then p, a new PageId object is created and the headerpage ID is stored in it.
- Using the p, we are pinning the page to pg1, a Page variable.
- Using the pinned page, the already existing BMPage is accessed, and its next page is set to INVALID.
- Now, available_space() function is used to check if the data is already present in the page or not.
- If data is already present, and still there is some available space in the page, then the number of data in the page, which is the count is stored in the integer variable count.
- If the position entered is greater than the number of data present in the page, then the position given is invalid and insert won't be performed.
- If not, setbit(position,0) is called and bit 0 is set in the given position successfully.
- If the data is present and there is no available space in the page, then a new BMPage (page1) and PageId (apage1) is created.
- The previous page, i.e. page, is used to set its next page as the new page which is created (page1) and page1 is used to set its previous page as page. apage1 is set as page1's page ID.
- By using apage1, a new Page variable, p2 is created so as to pin the page using apage1.
- Using p2, the already existing BMPage is accessed, and the its next page is set as INVALID.

- If there is space available, then just like the previous process using count, deletion is successfully implemented.
- After deletion is done, the page that is been currently pinned (modified) is unpinned.

destroyFile( ):

- In this function, if the header page is not null, i.e. the BitMapFile exists, then by using the header page ID through get_rootId() function, the existing BMPage is accessed.
- Using the BMPage variable (page) the dumpPage() function is used to destroy the page.
- The page is then freed and unpinned by using freePage(PageId) and unpinPage(PageId) respectively.
- The BitMapFile name in the database is then deleted by using the delete_file_entry function and the header page is set to null.

BMPage class:

This class is analogous to the HFPage class in the heap package of the minibase. It is present in the bitmap package.

Member Variables:

- DPFIXED, FREE_SPACE, PREV_PAGE, NEXT_PAGE, CUR_PAGE, availableMap are all the variables that are pre-defined in the minibase that is used here.
- COUNTER variable is defined here to help the tracking of the variable count.
- short count: this is used to keep track of the number of elements that have been inserted in a page
- short freeSpace: this keeps track of the number of bytes free in data[]
- PageId prevPage: this is a backward pointer to data page
- PageId nextPage: this is a forward pointer to data page
- PageId curPage: this gives the page number of the page

Member Functions:

getStartByte(): this is a function that returns the DPFIXED value.

getAvailableMap(): this function returns the availableMap value.

BMPage():

- A default constructor used to initialize a new BMPage.

- A new Page is declared and it's PageId is set.
- It is initialised by using the init() method present in this class.

BMPage( ): In this method the curPage is set to the passes parameter pid by using the method setCurPage(PageId pid) present in this class.

BMPage( ): This constructor is used to set the data by using the getpage() method on the page that is given as parameter.

available_space( ):

- This gets the value of the freeSpace variable from the data by using the FREE_SPACE variable.
- Then it returns the value of freeSpace.

getCount():

- This gets the value of the count variable from the data by using the COUNTER variable.
- Then it returns the value of count.

setCount(): Sets the COUNTER using the count variable.

dumpPage(): This method is used to dump the contents of a page.

empty():

- This is used to determine whether a page is empty.
- First a boolean variable isEmpty is set to false.
- The count variable is obtained from the data by using the COUNTER.
- If the count is zero then isEmpty is set to true.
- Returns the isEmpty variable.

init( )

- The data for the Page apage is obtained by using the getpage() method.
- count is set to 0 and it is updated in the COUNTER.
- pid of the curPage is set as the pid of the given pageNo and it is updated in the CUR_PAGE.
- pid of the nextPage and prevPage is set as INVALID_PAGE and it is updated in the NEXT_PAGE and PREV_PAGE.
- freeSpace is set as MAX_SPACE-DPFIXED and it is updated in FREESPACE.

openBMpage( ):

- This is a constructor of BMPage which opens an existed BMPage.
- It gets the data for the given page using the getpage() method.

getCurPage( ):

- This gets the pid value of the curPage from the data by using the CUR_PAGE variable.
- Then it returns the value of curPage.

getNextPage( ):

- This gets the pid value of the nextPage from the data by using the NEXT_PAGE variable.
- Then it returns the value of nextPage.

getPrevPage( ):

- This gets the pid value of the prevPage from the data by using the PREV_PAGE variable.
- Then it returns the value of prevPage.

setCurPage( ): This sets the pid value of the curPage as the pid of the given pageNo.

setNextPage( ): This sets the pid value of the nextPage as the pid of the given pageNo.

setPrevPage( ): This sets the pid value of the prevPage as the pid of the given pageNo.

getBMpageArray(): This returns the data in the BMPage.

writeBMpageArray( ): This sets the data in the BMPage to the byte array.

setBit( ):

- The data in the given position is set to the given bit.
- count variable is incremented.
- This is set by using the setCount( ) method.

ColumnIndexScan Class:

Column index scan file scans tuples based on index key. Directly the value of the tuple can be accessed using the key that is provided. This class is analogous to IndexScan.

<u>Member variables:</u>

- AttrType type: stores the values of attribute type.
- IndexType index: stores the type of index.

- IndexFileScan indScan: stores the values obtained from BTree_Scan by typecasting to BTFileScan type.
- String indName – a String which stores the file name
- String relName – a String which stores the name of the relation(name of the file)
- BTreeFile btf – used to open b-tree file
- heapfile f- used to open heapfile
- CondExpr[] Selects – stores the conditions that has to be applied while opening a file for scan.

Member functions:

getRelName() – returns the String value with name of the relation

getIndName() – returns the String value with input index name

getSelects() – returns the set of conditions

setSelects() – considers the values in selects(conditions)

getType() – returns the type of attribute.

setType() – sets the current object to this attribute type.

ColumnIndexScan():

- This is a parameterized constructor with parameters – index type, relation name, index name, attribute type, conditions, Boolean variable and size of string.
- Initialize the variables index_only, type1, short_size.
- Then we create an object of heap file.
- Use the indexType to check for the type of index file and perform respective scan operations.
- If the index type is btree - *BTree_scan*() is called.
- If the type is Bitmap, we read unique values from the column file and create a bitmap file for that value. This is obtained by declaring and initializing few temporary variables.
- Firstly, create a RID object. Using openScan function, open the heap file which is created in the above step.
- Call getColumnNumberFromColumname() method(defined in ColumnarFile) to get the column number.
- Create Hash set for Integer and String values as our data set has both string and integer values.

- Retrieve each value and check whether the value is already present in hashset or not. If the value is not present, add it to the hash set and create a bitmap file for it.
- Same would be the case for String values.

get_next():

- This function is used to retrieve the next tuple value in b tree index file.
- Only index value can also be returned based on the if condition.
- Header is set based on the type of tuple (integer value or string value).
- This function is also used to retrieve the next tuple value in bitmap index file.

setSlotvalue( ):

- This function is included in the HFPage class.
- It is checked if the value is of type IntegerValue or StringValue. The corresponding function in Convert class is used and the value is stored in given slot at the offset in the given byte array.

get_intvalue( ):

- This function is included in the HFPage class.
- The value at the given position in byte array is read using the getIntValue( ) function in Convert class.

get_strvalue( ):

- This function is included in the HFPage class.
- The value at the given position in byte array is read using the getStrValue( ) function in Convert class.

setCurPage_forGivenPosition( ) :

- This function is included in HFPage.
- This sets the value of the curPage to the page which contains the entry at the given position.
- This method takes in the position, column number and the columnarfile as its parameters.
- A columnar scan is performed on the file.
- It is scanned till it reaches the record in the given position.
- Once that record is found, the value of it is set as curPage value.

getvalue_forGivenPosition( ) :

- This function is included in HFPage.
- This gets the value at the given position in the given columnarfile.
- This method takes in the position, column number and the columnarfile as its parameters.
- A columnar scan is performed on the file.
- It is scanned till it reaches the record in the given position.
- Once that record is found, the value in the tuple is return.

ColumnarFileScan Class:

A class that is analogous to a file scan that helps in initializing scan. It scans a columnar file and returns tuples for which the given condition is satisfied. This is a part of Iterator Package.

Member Variables:

- String filename: Name of the file that has to be scanned
- Attrtype[] _ini: Attributes type
- short in1_len: Number of attributes
- short[] s_sizes: length of the string (If the input attribute is a string)
- int n_Out_Flds: Number of fields in the other tuple
- FldSpec[] perm_mat: Columns that have to be projected
- CondExpr[] OutputFilter: Condition expression
- Tuple tuple_ans: temporary tuple
- Tuple Jtuple: Final tuple to be returned

Member functions:

ColumnarFileScan():

- A constructor that initializes the columnar file scan object with the given attribute type, the number of attributes, the length of the string, the number of fields in the out tuple, the columns that have to be projected and the condition expression.
- Creates a new tuple object and sets the header page for it.
- Creates a heap file with the given filename.
- Opens the scan on the heap file created.

FldSpec[] show():

- Returns the perm_mat which contains the columns that has to be projected.

Tuple get_next():

- Creates a new RID and TID using the n_out_flds.

- It then starts the scan that was initialised in the constructor
- From the scan it gets the next tuple for that RID and sets the header for that tuple.
- It then calls the Eval function of the PredEval class.
- It passes the OutputFilter, tuple and the attribute of the tuple.
- It uses a parameter called rowpos which helps in finding the row number of the tuple.
- Thus at the end of every loop it increments the value of it.
- If the value returned by the Eval function is true, we will get the row number in which the condition was satisfied
- It then calls the Project function of the Projection class by passing the tuple, type of the tuple, Jtuple, columns to be projected and the number of fields.
- It then creates a HFPage currDirPage for the projection tuples that are needed and it contains a series of header page
- Each of it contains a pointer to a page and also the number of rows the page contains.
- Using the rowpos and the number of row value in the directory page, we iterate through the directory page
- We then try to find the page in which the condition became true.
- After finding the page in which the condition becomes true, we get into the page and iterate through it in order to find the tuple which is corresponding to the tuple that satisfied the condition.
- By doing the above jump we are able to skip lot of pages.
- Thus, it returns Jtuple that contains all the projected columns.

close():

- It closes the scan object that was initialised in the constructor.

PCounter Class:

A class that is used to initialize the read and write counter. This class is a part of DiskMgr Class.

Member Variables:

- Int rcounter: Used to have the number of reads.
- Int wcounter: Used to have the number of writes.

Member functions:

initialize(): It makes the value of rcounter and wcounter to 0.

readIncrement(): It increments the value of rcounter by 1.

writeIncrement(): It increments the value of wcounter by 1.

ColumnDB Class:

It is analogous to the DB class and extends it to have all the functionalities of the DB class with few modifications. It is a part of DiskMgr Class. It helps in opening, closing and destroying databases and also helps in reading and writing into a page.

Member Variables:

- RandomAccessFile fp: The random access file that is used for creating empty files and fill with zeros.
- int num_pages: Contains the number of pages that is required.
- String name: Contains the name of the file.

Member Functions:

open_DB():

- It gets the filename and deletes if there is any other file that is existing with same name.
- It then creates a random accesses it file and fills it with zeroes with the help of the num_pages variable.
- It creates a new page, pageid and pins them.
- It creates the first page of the DB with the help of the newly created page.
- It then sets the bit and the PCounter is initialised.

read_page():

- It takes in the page and the pageid as input and in the random access file it seeks to the page that is need using the pageid.
- It then gets the page into a byte array and reads it.
- After reading the page, the read counter is incremented using the readIncrement function.

write_page():

- It takes in the page and the pageid as input and in the random access file it seeks to the page that is need using the pageid.
- It then gets the page into a byte array and writes into it.
- After writing into the page, the write counter is incremented using the writeIncrement function.

Main Class:

Class where execution of the code starts. As a command line argument we are giving input in the following format

DATAFILENAME COLUMNDBNAME COLUMNARFILENAME NUMCOLUMNS

It is a menu driven program and have 5 option:

CSE 510: Columnar Minibase

Menu:

1. Batch Insert Program
2. Index Program
3. Query Program
4. Delete Query Program
5. Quit Program

Enter Option from above:

Depending upon the input value corresponding function is called

Option 1: Batch Insert Program

- We are calling batchInsert() function that reads the header of the given input file and parse the given input i.e. determines the data type of the columns, size (if string) and calls the columnar file constructor to create columnar file object.
- Once the columnar file object is created the file is further read and data is inserted into the corresponding column heapfile.
- Further disk reads and writes are calculated in the complete procedure.
- After successful insertion of records, using tuple scan class all the inserted records were printed and checked whether they were inserted correctly or not.
- For this tuple scan object for created.
- Tuple metadata was set and using this tuple scan class getNext() function records were fetched and displayed using the predefined Tuple class functions.

Option 2: Index Program

- This option creates the index on the given column name.
- Input is taken from user which contains
- Once input is taken. It open the corresponding heap file using the columnar metadata.
- Disk reads and writes were initialized
- Scan is opened on the corresponding column heap file.
- Further BTreeFile object is created.
- Tuples are read from the heap files and index file is created using the values.
- After successful creation of BTree index, root is displayed along with the leaf pages.

If option is chosen as bitmap

- Once input is taken. It open the corresponding heap file using the columnar metadata.
- Disk reads and writes were initialized
- Scan is opened on the corresponding column heap file.
- Maintained 2 hash set in order to store the unique integer and string values. If the value is not found in the hash set a new BitMapFile corresponding to it is created.
- The name of the BitMapFile name is given as concatenation of columnarfilename, column number and value.
- Tuples are read from the heap files and index file is created using the values.
- After successful creation of BitMapFile pages are displayed.

Option 3: Query Program

- When chosen this as option query program gets executed
- Input is given in the following format

  query COLUMNDBNAME COLUMNARFILENAME [TARGETCOLUMNNAMES]

  VALUECONSTRAINT NUMBUF ACCESSTYPE

- The input is further parsed to find the index columns and the access type and the values required for the query execution using standard JAVA string manipulation.
- Further depending upon the access type mentioned query is executed and results are fetched.

1. If the access type chosen is BTREE,
   - the index file corresponding to the given column is opened.
   - index scan is initiated on the the opened index file.
   - Keys fetched from the index file is compared to the given value and the value along with its position is printed

2. If AccessType chosen is BITMAP,
    - The bitmap index required is opened by using the filename given as concatenation of columnarfilename, column number and value.
    - The number of positions where the value matches is found using the getpositions() and the number of matching positions is found using getCount().
    - The getvalue_forGivenPosition( ) is used to find values in the target columns for the positions we get in the position array.

3. If the AccessType chosen is COLUMNSCAN,
    - The parameters for the ColumnarFileScan is computed and sent as input to its constructor and get the object for the ColumnarFileScan.
    - Then we iterate through the columnar file with the help of get_next function and get all the columns that are given for the projection.
    - On getting each of the tuple, we print the respective field that we want according to the attribute type.

4. If the accesstype chosen is FILESCAN,
    - The parameters for the ColumnarFileScan is computed and sent as input to its constructor and get the object for the ColumnarFileScan.
    - Then we iterate through the columnar file with the help of get_next function which contains all the tuples that are required.
    - We print each of the tuples with the respective field.

Option 4: Delete Query

Same methodology was used in here for parsing. After successfully parsing the data, the markTupleDeleted() or purgeAllDeletedTuples() is called.

- Input is given in the following format

  query COLUMNDBNAME COLUMNARFILENAME [TARGETCOLUMNNAMES]

  VALUECONSTRAINT NUMBUF ACCESSTYPE
- The input is further parsed to find the index columns and the access type and the values required for the query execution using standard JAVA string manipulation.
- Further depending upon the access type mentioned query is executed and results are fetched.

1. If the access type chosen is BTREE,
    - the index file corresponding to the given column is opened.
    - index scan is initiated on the the opened index file.
    - Keys fetched from the index file is compared to the given value and the value along with its position is printed.
    - The matching values are deleted.
2. If AccessType chosen is BITMAP,
    - The bitmap index required is opened by using the filename given as concatenation of columnarfilename, column number and value.
    - The number of positions where the value matches is found using the getpositions() and the number of matching positions is found using getCount().
    - The values in the target columns are deleted using columnar file delete functions.
3. If the AccessType chosen is COLUMNSCAN,
    - The parameters for the ColumnarFileScan is computed and sent as input to its constructor and get the object for the ColumnarFileScan.
    - Then we iterate through the columnar file with the help of get_next function and get all the columns that are given for the projection.
    - On getting each of the tuple, values in the target columns are deleted using columnar file delete functions
4. If the accesstype chosen is FILESCAN,
    - The parameters for the ColumnarFileScan is computed and sent as input to its constructor and get the object for the ColumnarFileScan.
    - Then we iterate through the columnar file with the help of get_next function which contains all the tuples that are required.
    - values in the target columns are deleted using columnar file delete functions.

Future Work:

- Value class can be extended further to support other data types also.

Conclusion:

We have used various modules of the Relational Database Management Systems that were implemented by the MiniBase. The purpose of these modules is to implement a column-oriented DBMS. All the programs for batchinsert, index, query and delete_query were implemented.

References:

1. https://docs.oracle.com/cd/B28359_01/server.111/b28313/indexes.html
2. http://db.csail.mit.edu/projects/cstore/abadi-sigmod08.pdf
3. https://en.wikipedia.org/wiki/Column-oriented_DBMS
4. The Minibase Homepage *http://research.cs.wisc.edu/coral/minibase/minibase.html*
5. http://www.cs.carleton.edu/faculty/dmusican/cs334f07/proj1.html
6. http://www.cs.colostate.edu/~iray/teach/cs430/BufMgr.pdf
7. https://www.eecs.yorku.ca/course_archive/2003-04/W/4411/proj/smj/
8. http://www.cs.sfu.ca/CourseCentral/354/zaiane/material/notes/Chap7/node11.html
9. https://www.tutorialcup.com/dbms/index.html
10. https://web.stanford.edu/class/cs346/2015/notes/Lecture_One.pdf

Appendix:

Division of Work:

Sucharitha Rumesh:

BitMapFile

Supriyaa Damodaraswamy:

BMPage

Extending HFPage with setCurPage_forGivenPosition

Spoorthi Karnati:

ColumnIndexScan

Naga Subrahmanya Sai Tarun Malladi:

ColumnarFileScan

Delete_query

Hariharan Adhithya Venkatraman:

ColumnDB

PCounter

Query for Columnscan and Filescan

Thiviyakalyani Navaneethan:

BM

BitMapHeaderPage

Added some functions in HFPage

Query for bitmap
Documentation Compiling

Ashish Agarwal:

ValueClass
TID
ColumnarFile
TupleScan
Main Class
Batch Insert
Index Program - BTree Index
BitMap Index (Unique value - used hash set)
Query - BTree