?

§5 The Access Method (AM) Layer

# § 6. The Front End (FE) Layer

For this part of the project, you are to implement the front-end of MiniRel. The front-end layer consists of (1) database utility functions, (2) data definition language (DDL) functions, and (3) data manipulation language (DML) and query functions. You also have to implement the catalog relations.

## § 6.1. Catalogs

One of the nice things about a relational database system is that the internal schema information that describes what relations exist in the database, the format of their attributes, and what attributes have indices on them, can also be stored in relations, called *catalogs*. Thus, the catalogs can be manipulated using the access methods that you implemented for the HF and AM layers.

There are two essential catalogs: *relcat* and *attrcat*. These two relations should be created when a database is created. They should then be placed in a subdirectory, which is also created when the database is created, so that different databases can have their own schemas and database files in separate directories. The `relcat` relation will be used to keep track of all of the relations in the database. Each tuple in the `relcat` relation should have a structure similar to `RELDESCTYPE` defined in the `catalog.h` file in public directory. The `attrcat` relation will be used to keep track of all of the attributes of all of the relations in the database. Each tuple in the `attrcat` relation should have a structure similar to `ATTRDESCTYPE` defined in the `catalog.h` file.

Note that both the catalog relations are structured as a regular relation in the database. When the catalogs are created, `relcat` should have two tuples for the catalog relations and `attrcat` should have tuples for all the attributes of both `relcat` and `attrcat`. Also note that the two names, `relcat` and `attrcat`, are reserved for the catalog relations. This means you are not allowed to create a relation named either `relcat` or `attrcat`.

## § 6.2. Database Utility Functions

- **void DBcreate(char *dbname)**
  This function creates a new database called `dbname`. A database needs its own directory (preferably named `dbname`), which can be created by executing a Unix system call `mkdir`. That is where all the database files as well as catalogs will be stored. The catalog relations `relcat` and `attrcat` must be created and stored in the directory.

- **void DBdestroy(char *dbname)**
  This function destroys database `dbname` by removing the directory of the database and everything in the directory.

- **void DBconnect(char *dbname)**
  This function opens and connects to database `dbname` by changing to the directory of the database (by a Unix system call `chdir`) and opening the catalog

relations. The catalog relations must be kept open while you are connected to the database.

- void DBclose(char *dbname)
  This function closes database dbnameby closing the catalog relations and changing back to the parent directory. The database must have been opened by the DBconnect function.

# § 6.3. Data Definition Language (DDL) Functions

Once MiniRel is invoked, one can ask it to execute a command to create or destroy a relation and build or drop an index. Such a command is part of the DDL (data definition language) of the system, whereas queries and updates are part of the DML (data manipulation language) of the system. The DDL and other utility functions you will implement are described in this section. The prototypes of the functions and other useful definitions are provided in the fe.h in the public directory.

- int CreateTable(…)

```
char       *relName;       /* name of relation to create   */
int        numAttrs;       /* number of attributes         */
ATTR_DESCR attrs[];        /* attribute descriptors        */
char       *primAttrName;  /* primary key attribute        */
```

  This function creates a relation as specified in the parameters. Relation and attribute names can be at most MAXNAME characters long. As each relation is created, CreateTable will have to update the catalog information appropriately, by adding one tuple to the relcat relation describing the relation and then add one tuple to the attrcat relation for each attribute that it has. After updating the catalogs, CreateTable should call HF_CreateFile() to create a file to hold tuples of the relation. You will have to calculate the tuple length yourself by summing the lengths of each attribute.

  The ATTR_DESCR type is defined in fe.h and used to represent attributes defining the relation. The primAttrName parameter must be set to NULL because we do *not enforce primary key integrity*. This function returns FEE_OK if it succeeds, and an FEE error code otherwise.

- int DestroyTable(char *relName)

```
char    *relName;          /* relation name */
```

  This function destroys relation relName and the file that contains the relation by calling HF_DestroyFile(). It should also destroy any index built on the relation. The catalog relations need to be updated appropriately. This function returns FEE_OK if it succeeds, and an FEE error code otherwise.

- int BuildIndex(char *relName, char *attrName)

```
char    *relName,         /* relation name                */
char    *attrName         /* name of attr to be indexed   */
```

  This function creates an index on attribute attrName of relation relname by calling AM_CreateIndex(). Existing tuples of the relation must be inserted into

the index, once the index is created. The catalog relations need to be updated appropriately. This function returns FEE_OK if it succeeds, and an FEE error code otherwise.

- int DropIndex(char *relName, char *attrName)

```
char    *relName,       /* relation name                */
char    *attrName       /* name of indexed attribute    */
```

This function drops indices from relation relname. If attrname is non-null, then it is the name of the attribute whose index is to be dropped (if it exists). If attrname is NULL, then all indices, which can be found by examining the catalogs, are dropped from the relation. This function returns FEE_OK if it succeeds, and an FEE error code otherwise.

- int LoadTable(char *relName, char *fileName)

```
char    *relName,       /* name of target relation      */
char    *fileName       /* file containing tuples       */
```

This function loads a group of (binary) tuples from input file fileName into relation relName. Any index on the relation is updated appropriately. Note that the input file fielName is a standard Unix binary file and has nothing to do with the HF or AM layer file format. This function returns FEE_OK if it succeeds, and an FEE error code otherwise.

- int HelpTable(char *relName)

```
char    *relName,       /* relation name */
```

If relName is NULL, this function prints the content of the relcat catalog. Otherwise, it lists the name, type, length, offset of each attribute of the specified relation, together with any other information you feel is useful. This function returns FEE_OK if it succeeds, and an FEE error code otherwise.

- int PrintTable(char *relName)

```
char    *relName,       /* relation name */
```

This function prints the contents of the specified relation. The printing should be a table with each row representing one tuple of the relation. The order of the attributes in the table should be the same as the order in which the attributes were presented at the creation time. This function returns FEE_OK if it succeeds, and an FEE error code otherwise.

## § 6.4. Data Manipulation Language (DML) and Query Functions

For this part of the project, you are to implement functions for the DML and query operators such as insert, delete, select, and join. You will be able to use these functions to retrieve and modify data stored in a database. Note that the select and join operators may be followed by a projection. The function prototypes are defined in the fe.h file in the public directory.

- int Insert(char *relName, int numAttrs, ATTR_VAL values[])

```
char     *relName;        /* target relation name       */
int      numAttrs;        /* number of attribute values */
ATTR_VAL values[];        /* attribute values           */
```

This function inserts a tuple represented by values[] into relation relName. The ATTR_VAL type is defined in fe.h and is used to represent an attribute-value pair. The attributes may not be given in the order they are structured in the relation, so make sure you find the appropriate place for each one. If no value is specified for an attribute, you should reject the insertion and return an error code. All the associated indices should be updated when the insertion is done. This function returns FEE_OK if it succeeds, and an FE error code otherwise.

- int Delete(…)

```
char    *relName;         /* target relation name         */
char    *selAttr;         /* name of selection attribute  */
int     op;               /* comparison operator          */
int     valType;          /* type of comparison value     */
int     valLength;        /* length if type = STRING_TYPE */
void    *value;           /* comparison value             */
```

This function deletes tuples that satisfy a deletion criterion from relation relname. The deletion criterion is specified by the rest of the parameters. If selAttr is a NULL pointer, then all tuples are deleted. All the associated indices should be updated when the deletion is done. This function returns FEE_OK if it succeeds, and an FE error code otherwise.

- int Select(…)

```
char    *srcRelName;       /* source relation name         */
char    *selAttr;          /* name of selected attribute   */
int     op;                /* comparison operator          */
int     valType;           /* type of comparison value     */
int     valLength;         /* length if type = STRING_TYPE */
void    *value;            /* comparison value             */
int     numProjAttrs;      /* number of attrs to print     */
char    *projAttrs[];      /* names of attrs of print      */
char    *resRelName;       /* result relation name         */
```

This function inserts tuples satisfying a selection criterion into resRelName, which may or may not exist in the database. If the result relation does not exist in the database, it should be created. The source and result relations must be distinct. If resRelName is set to NULL, the query results should be directed to *stdout*.

The selection criterion is determined by selAttr, op, valType, valLength, and value parameters. If selAttr is a NULL pointer, then all tuples are selected. The attributes to project are determined by numProjAttrs and projAttrs[]. You should process the projection on the specified columns while the selection is being processed. You do not have to worry about eliminating duplicates. This function returns FEE_OK if it succeeds, and an FE error code otherwise.

- int Join(…)

```
REL_ATTR *joinAttr1;       /* join attribute #1          */
int      op;               /* comparison operator        */
REL_ATTR *joinAttr2;       /* join attribute #2          */
int      numProjAttrs      /* number of attrs to print   */
REL_ATTR projAttrs[];      /* names of attrs to print    */
char     *resRelName;      /* result relation name       */
```

This function performs a join between two relations, where the joining attributes are represented by `joinAttr1` and `joinAttr2`. The `REL_ATTR` type is defined in `fe.h` and is used to represent a relation-attribute pair. The attributes to project from the join are specified in `projAttrs[]` and the result of the join will be inserted into `resRelName`. The output relation `resRelName` may or may not exist in the database. If the output relation does not exist in the database, it should be created. The two input relations to be joined must be distinct (*i.e.*, no self-join) and the output relation must be also distinct from the input relations. If `resRelName` is set to NULL, the query results should be directed to *stdout*. This function returns `FEE_OK` if it succeeds, and an FE error code otherwise.

# § 6.5. FE Miscellaneous Functions

- void FE_Init(void)
  This function is used to initialize the underlying layers. This can be as simple as invoking just `AM_Init()`, provided that the underlying layers are initialized by that in proper order. This function takes no parameters and produces no return value.

- void FE_PrintError(char *errString)

```
char    *errString;        /* pointer to an error message */
```

  This routine writes the string `errString` onto stderr, and then writes the last error message produced by the FE layer onto stderr as well. This routine will make use of a global integer value, `FE_errno`, in order to keep track of the most recent error; this error code should be appropriately set in each of the other FE layer routines. This routine has no return value of its own.

# § 6.6. Implementation Notes

## § 6.6.1. Input file format for LoadTable

The input file `fileName` of the `LoadTable` function is a standard Unix binary file without any meta data in it. This file contains a number of binary tuples in the consistent form for the relation which the tuples will be loaded into. There is no need to check that the input file is consistent with the schema of the relation. The attributes and tuples in the input file are *byte-aligned*. There is no padding between any consecutive attributes and between any consecutive tuples in the input file. (Consider a C structure compiled with a `#pragma pack(1)` directive.)

## § 6.6.2. Bulk loading of B$^+$-tree

The AM layer does not support the bulk loading operation of B$^+$-tree. When an index is created, if the base relation is not empty, new index entries for all existing tuples must be created and added to the index by invoking `AM_InsertEntry` function.

However, inserting many entries individually may take a long time. Although it is not required, You may want to consider adding a bulk loading function to your AM layer.

## § 6.6.3. One database connection at a time

When closing a database, the database must have been opened by the DBconnect function. When the DBclose function is invoked with a dbname parameter, you may want to check whether dbname matches the name of the currently open database. Alternatively, you can simply ignore the dbname parameter and close the currently open database, because we assume that MiniRel lets you connect to just one database at a time.

---