**Programming Project #2: Database Files and Indexing**

**CS-6360 Database Design**

**Instructor: Chris Irwin Davis**

**Team Members: Anki Chauhan, Arihant Chhajed, Qi Gao, Nilkumar Patel, Meghna Kurup**

**Introduction:**

The main aim of the project was to implement a basic database engine that is based on a simplified file-per-table variation on the SQLite file format. The variation created is called Davisbase and in this approach, each database table is a separate file with logical subdivisions called pages. The implementation was done using java.

**Commands and Basic Working:**

As per the requirements we have implemented the DDL, DML and VDL commands in our Database Engine. Our database engine uses ‘davissql>’ as a prompt. The DDL or Data Definition Language commands implemented are:

* Show tables
* Create Index
* Create table
* Drop table

Show tables will display a list of tables in DavisBase. Create table is used to create a new table and Drop table is used to drop an existing table. Sample code for each command is given below.

* Show tables:
  + davissql>Show tables;
* Create Index:
  + Davissql> create index idx on abc(“Davis”);
* Create table:
  + davissql>create table ABC (id INT [Primary Key], name TEXT[not null], sex TEXT, age INT);
* Drop table:
  + davissql>Drop table ABC;

The DML or Data Manipulation Language commands implemented in our DavisBase Engine are:

* Insert
* Delete
* Update

Insert is used to insert a new record into a table file. The Delete command deletes a record from a table file. The update command is used to modify a record that is already present in the table. The result displayed on the terminal on executing these commands are the rows affected after the execution of the command. Sample code for each command is given below.

* Insert:
  + Insert into ABC (id,name,sex,age) values (1,“David”,”Male”,20);

The DQL or the Data Query Language implemented in the Engine is:

* Select-From-Where

This simple Select-From-Query is used to query rows from the table. Our implementation does not include nested Where clauses, Joins, Order By and Group By. The Where condition applies to multiple records.The results are the rows affected and the columns satisfying the Where clause if present. Sample code for Select -From-Where is given below.

* Select \* from ABC;
  + This will return all the rows and columns stored in the table ABC.
* Select \* from ABC where id=2;
  + This will return the row and columns for where id=2.

**Internal Working:**

In our implementation when each table is created a new file is created. The same thing happens when we create an index. The table file created has the format ‘tablename.tbl’. Each index file is stored in a file whose name is ‘tablename\_columnname.ndx’.

Each file has logical sections called pages. The page size used in the implementation is 512.

To locate an element within a file we use offset. They represent the number of bytes from a given reference point and are expressed as non-negative integers. A file offset will give the location of an element within the file. Page offset will give the location within a page and it is the number of bytes from the beginning of the page.

Table Files are not required to have any Index Files, i.e. they may have zero-to-many indexes. However, every Index File must have exactly one Table File that it indexes.

Our implementation uses rowid to internally identify records and each record can be uniquely identified by its rowid.

All DavisBase indexes provide access paths to records based only on single columns. Our implementation does not support multi column indexing. Index Files has indexing information which provide an access path to a related Table File.

**Implementation Details**

The source files contains service files which include DavisbaseManager, DavisbaseValidator and DavisbaseExecutor. The DavisManager as the name suggests is like a manager code which is responsible for what the user can see. It provides a flow and acts as a link between the validator and executor. The Validator validates the commands of the DavisBase engine. The Executor as the name suggests is responsible for the execution of the commands.

The DavisBase model files consist of the error and exception files. It returns the errors during the program run. The DavisBase model QueryType files consist of the running for each command which is used in the Executor file.

The Davisbase utils consist of the utils file. These include datatype and constant files. It also includes the DavisBaseFileHandler and DavisBaseCatalogHandler file. The FileHandler is used to read or write data from .tbl files. The DavisBaseCatalogHandler class is used to read/write the Database catalog files. Since the Catalog files are also treated as system tables they will be read or written using DavisBaseFileHandler.

The Davisbase model PageComponent is for the pages. It consists of the PageHeader class to represent the page header. The PageType class represents page type. Whether the page is an index node or leaf or a table node or leaf. The cell interface used in B and B+ tree. The CellHeader class represents the CellHeader, the CellPayload class represents the Cell payload. The InternalColumns class represents the internal columns like dataType or isPrimary. The LeafCell class and NonLeafCell classes represent the leaf and non-leaf cells in the B tree.

Along with the above files we have the data files which include the tables created and the table for columns.

For indexing the implementation is done with separate CellHeader class and CellPayload class. InternalColumns class allows to set and get the index value. Indexes have separate offsets from tables. The QueryType files for each command also has the implementations for accommodating the use of an index.

The DavisBase architecture borrows from SQLite, MySQL, and PostgreSQL but not completely. DavisBase is not completely ACID compliant. Also, DavisBase does not have a good mechanism for recovery or failed transactions. Thus, the implementation is a subset of the SQL specification.