Universidad Tecnológica Centroamericana

Teoría Bases de Datos II – CCC304

Primer Semestre, Segundo Periodo, 2016

Proyecto Segundo Parcial

Este proyecto tiene como objetivo desarrollar las operaciones del DBMS BruinBase que se dan a continuación:

Parte A

Se debe de implementar el comando SQL bulk load similar al comando LOAD DATA que se usa en MySQL. Para Bruinbase, la sintaxis para cargar datos en una tabla es:

LOAD nombre_tabla FROM ‘nombreArchivo’ [WITH INDEX]

El commando crea la table con el nombre nombre_tabla y carga los pares (clave, valor) desde el archivo con nombre nombreArchivo. La clave es un entero y el valor es una cada que este entre doble comillas

Por ejemplo

1,"valor 1"
2,"valor 2"
...

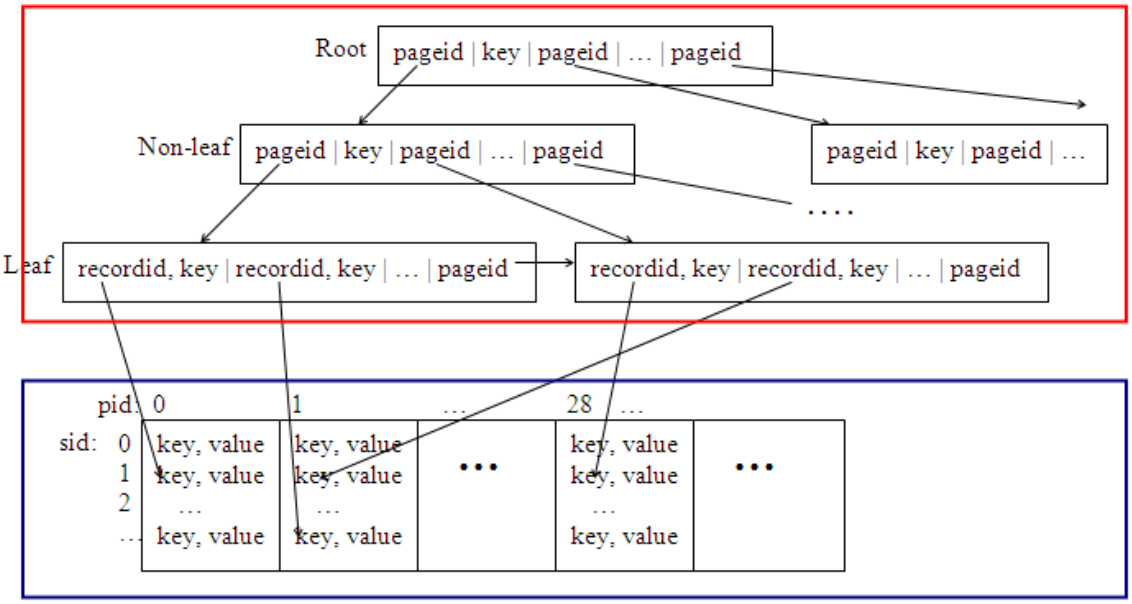
Note que cuando el usuario usa el comando LOAD, Bruinbase invoca la función SqlEngine::load(table, loadfile, index). Usted debe de implementar esta función. El parámetro index is TRUE si se usar la opción WITH INDEX.

Para crear la tuplas de la tabla use la clase RecordFile. Cuando se crear una nueva tabla, el RecordFile debe ser llamado como nombre_tabla+”.tbl”. Si la tabla ya existe, se le agregan las tuplas del archivo a la tabla. Para pueba, usa el archivo movies.del para cargar la tabla movies en otra tabla.

PARTE B

Agregar el arbol B+ a la base de datos. Cada nodo del árbol corresponde a una página en PageFile. A pageID que es almacenado en un nodo interior es una referencia a un nodo hijo. El último pageID en una hoja es una referencia al hermano siguiente. El recordID almacenado en una hoja es una referencia a un RecordFile.

Estructura del árbol B+



Los tres métodos más importantes de BTreeIndex son los siguientes:

```
/**
   * Insert (key, RecordId) pair to the index.
   * @param key[IN] the key for the value inserted into the index
   * @param rid[IN] the RecordId for the record being inserted into the
index
   * @return error code. 0 if no error
   */

RC insert(int key, const RecordId& rid);
/**
   * Run the standard B+Tree key search algorithm and identify the
   * leaf node where searchKey may exist. If an index entry with
   * searchKey exists in the leaf node, set IndexCursor to its location
```

```
   * (i.e., IndexCursor.pid = PageId of the leaf node, and
   * IndexCursor.eid = the searchKey index entry number.) and return 0.
   * If not, set IndexCursor.pid = PageId of the leaf node and
   * IndexCursor.eid = the index entry immediately after the largest
   * index key that is smaller than searchKey, and return the error
   * code RC_NO_SUCH_RECORD.
   * Using the returned "IndexCursor", you will have to call readForward()
   * to retrieve the actual (key, rid) pair from the index.
   * @param key[IN] the key to find
   * @param cursor[OUT] the cursor pointing to the index entry with
   *                    searchKey or immediately behind the largest key
   *                    smaller than searchKey.
   * @return 0 if searchKey is found. Othewise, an error code
   */
  RC locate(int searchKey, IndexCursor& cursor);

  /**
   * Read the (key, rid) pair at the location specified by the IndexCursor,
   * and move foward the cursor to the next entry.
   * @param cursor[IN/OUT] the cursor pointing to an leaf-node index entry
in the b+tree
   * @param key[OUT] the key stored at the index cursor location
   * @param rid[OUT] the RecordId stored at the index cursor location
   * @return error code. 0 if no error
   */
  RC readForward(IndexCursor& cursor, int& key, RecordId& rid);
```

Además, se deben implementar las clases BTLeafNode y BTNonLeafNode que están en BTreeNode.h

Para usar el índice usa la instrucción

```
LOAD tablename FROM 'filename'  WITH INDEX
```

Modifique SqlEngine

En la parte final del Proyecto, se debe modificar el SQL ENGINE para hacer uso del código de indexación. Se debe de aumentar la funcion SqlEngine::load() para generar un indice para la table y la funcion SqlEngine::select() para hacer uso de su índice en tiempo de ejecución del query

- **SqlEngine::load(const string& table, const string& loadfile, bool index)**
  Si se modifica esta function tal que el tercer parametro index es verdadero, Bruinbase crea el índice con el árbol B+ correspondiente. El archivo índice debe llamarse 'tblname.idx' y debe aparecer en el directorio actual.
-
- **SqlEngine::select (int attr, const string &table, const vector<SelCond> &conds)**
  La función `select()` es llamada cuando el usuario ejecuta el comando SELECT. El parámetro `attr` (`attr=1` significa "key" attribute, `attr=2` significa "value" attribute, `attr=3` significa "*", and `attr=4` significa "COUNT(*)"). El nombre de la table es pasada en el parametro table. Las condiciones listadas en WHERE son pasadas con el parámetro conds, el cual es un vector de SelCond

```
struct SelCond {
  int attr;     // 1 "key" attribute. 2 "value" attribute.
  enum Comparator { EQ, NE, LT, GT, LE, GE } comp;
  char* value;    // the value to compare
};
```

  Por ejemplo, para la condición como "key > 10", `SqlEngine` pasa a `SelCond` con "attr = 1, comp = GT, y value = '10'".

  La implementación actual de `select()` realiza un escaneo de table para resolver el query.

  La tarea a resolver para el `select()` es la siguiente:

  - Si el SELECT tiene una o mas condiciones en la columna clave y si la table tiene un arbol B+, entonces se debe de usar el arbol B+ para ayudar obtener la respuesta del query

- Si hay un condicion de desigualdad en la clave
- Si el query especifica un rango (like key >= 10 and key < 100)

En el archive Proyecto-test.zip se incluyen 5 archivos con datos de muestra, un script de Shell, un script SQL y el resultado esperado. Las pruebas incrementan la complejidad del acceso al árbol B+.