CS525-F24-G17

CS 525: Advance Database Organization

PROGRAMMING ASSIGNMENT 3: RECORD MANAGER

Group_17 Members:

• Venkata Naga Lakshmi Sai Snigdha Sri Jata - A20560684 - 25% Contributed • Sharan Rama Prakash Shenoy - A20560683 - 25% Contributed • Adarsh Chidirala - A20561069 - 25% Contributed • Ajay Kumar Choudary Koneti - A20563634 - 25% Contributed

---

README- Record Manger

This README provides an overview of the Record Manager code, which is divided into five main sections: Tables and Manager, handling records in a table, scans, dealing with schemas, and dealing with records and attribute values. Each section explains the purpose and functionality of the relevant functions and methods.

Optional Extensions: We have implemented an Interactive Interface extension that allows you to define new tables, insert, update, and delete tuples, and execute scans. This implementation provides a menu-based command-line interface (CLI).

The Procedure to Run the Code:

Step 1: Navigate to the new branch created "Assignment3" branch in Git-Hub and download the necessary .zip file.

Step 2: Run "make clean" to remove any previously compiled files.

Step 3: Use the "make" command to compile the program by running the Makefile.

Step 4: Execute the test case 3_1 by running "./test_assign3_1".

Step 5: Execute the test expressions by running "./test_expr".

Step 6: Execute the test Interactive Interface by running "./Interface".

Step 7: Once again, use "make clean" to remove generated executable files, and repeat steps 3 and 5 to rerun the assignment.

Features To Do

1. "Table and Manager:" Implement functions and methods for managing tables, including creating, deleting, and altering tables.
2. "Record Management:" Handle records in a table, including inserting, deleting, and updating records.
3. "Scans:" Perform scans on tables, including sequential scans and index scans.
4. "Schema Management:" Manage the schema of a table, including creating, altering, and dropping schemas.
5. "Attribute Values:" Handle attribute values in records, including retrieving, modifying, and comparing attribute values.

Record Manager

This README provides a detailed guide to the Record Manager, including the process of managing tables, handling records, dealing with schemas, and memory management operations. The Record Manager is essential for performing various database operations such as creating tables, inserting records, and manipulating schemas.

Part 1: Record Manager Initialization

This section covers the initialization and shutdown of the Record Manager, alongside basic table operations.

```
RC initRecordManager(void *mgmtData)
```

- **Description**: This function initializes the Record Manager by allocating memory and setting up the internal structures required for managing records and tables. It ensures that the system is ready for future table-related operations.
- **Purpose**: Prepares the Record Manager for further operations by initializing required resources.
- **Returns**: `RC_OK` on success, error code otherwise.

`RC shutdownRecordManager()`

- **Description**: This function gracefully shuts down the Record Manager, deallocating any memory and resources used. It ensures that no memory leaks occur by releasing the memory associated with the schema and setting it to `NULL`.
- **Purpose**: To cleanly shut down and free resources.
- **Returns**: `RC_OK` on success, error code otherwise.

`RC createTable(char *name, Schema *schema)`

- **Description**: This function creates a new table, initializing the table's schema and metadata. It opens or creates a page file for the table and sets up the buffer pool for efficient access. Additionally, the table's schema and attributes are serialized and stored for later use.
- **Purpose**: Creates a new table with a specified schema, setting up the storage and buffer management system for it.
- **Returns**: `RC_OK` on success, `RC_ERROR` on failure.

Part 2: Managing Tables

This section describes functions to open, close, and delete tables.

`RC openTable(RM_TableData *rel, char *name)`

- **Description**: Opens an existing table by its name. If the file exists, it initializes the table metadata (such as schema and file handle) and loads the table for further operations. If the file does not exist, it returns an error.
- **Purpose**: To load an existing table into memory for reading or modification.
- **Returns**: `RC_OK` if the table is successfully opened, `RC_FILE_NOT_FOUND` if the file doesn't exist.

`RC closeTable(RM_TableData *rel)`

- **Description**: Closes a previously opened table, ensuring that all the resources used by the table (e.g., file handles, buffer pool) are correctly released.
- **Purpose**: Safely closes the table, freeing up resources for future operations.
- **Returns**: `RC_OK` on successful closure, error code if any issues occur.

`RC deleteTable(char *name)`

- **Description**: This function deletes the specified table by removing the page file associated with it. It checks if the table file exists and deletes it using the appropriate file system function.
- **Purpose**: Permanently removes a table from the database by deleting its page file.
- **Returns**: `RC_OK` if the file is successfully deleted, `RC_FILE_NOT_FOUND` if the file doesn't exist.

Part 3: Handling Records

This section explains how to insert, delete, update, and retrieve records from a table.

`RC insertRecord(RM_TableData *rel, Record *record)`

- **Description**: Inserts a new record into the table. It calculates the appropriate page and slot to store the record, copies the data into the table, and updates the record count.
- **Purpose**: Adds a new record to the table and updates table metadata.
- **Returns**: `RC_OK` if the record is successfully inserted.

`RC deleteRecord(RM_TableData *rel, RID id)`

- **Description**: Deletes a record from the table based on its unique Record ID (RID). It frees the space used by the record and updates the metadata, such as the record count.
- **Purpose**: Removes a specific record from the table by clearing its data and releasing its slot.
- **Returns**: `RC_OK` if the record is successfully deleted.

`RC updateRecord(RM_TableData *rel, Record *record)`

- **Description**: Updates an existing record with new data. The function locates the record using its ID, replaces the old data with the new one, and updates the page in the buffer.
- **Purpose**: Modifies the content of an existing record in the table.
- **Returns**: `RC_OK` if the update is successful.

`RC getRecord(RM_TableData *rel, RID id, Record *record)`

- **Description**: Retrieves a record from the table by its Record ID (RID). It reads the data from the correct page and slot and returns it in the provided record structure.
- **Purpose**: To retrieve a specific record from the table for viewing or modification.
- **Returns**: `RC_OK` if the record is successfully retrieved.

Part 4: Scanning Records

This section explains how to perform scans over records, applying conditions and retrieving matching results.

`RC startScan(RM_TableData *rel, RM_ScanHandle *scan, Expr *cond)`

- **Description**: Initializes a scan on the table, allowing records to be sequentially retrieved based on a condition. The scan starts at the beginning of the table, and each record is evaluated against the given condition.
- **Purpose**: To start scanning a table for records that meet a specified condition.
- **Returns**: `RC_OK` if the scan is successfully started.

`RC next(RM_ScanHandle *scan, Record *record)`

- **Description**: Retrieves the next record in the scan that satisfies the condition. If no records are left or no record matches the condition, an appropriate message is returned.
- **Purpose**: To fetch the next matching record in the scan.
- **Returns**: `RC_OK` if a matching record is found, `RC_RM_NO_MORE_TUPLES` if no more records are available.

`RC closeScan(RM_ScanHandle *scan)`

- **Description**: Closes the scan and frees any resources used during the scan. This ensures that scans are properly terminated and can be reused later.
- **Purpose**: To close and clean up after a scan operation.
- **Returns**: `RC_OK` on successful closure.

Part 5: Schema Management

This section covers the creation, management, and deletion of schemas that define the structure of tables and records.

`int getRecordSize(Schema *schema)`

- **Description**: Calculates the size of a record based on the attributes defined in the schema. The record size is determined by the data types and lengths of the attributes.
- **Purpose**: To determine the total size of a record in bytes, based on the schema.
- **Returns**: The size of the record in bytes.

`Schema *createSchema(int numAttr, char **attrNames, DataType *dataTypes, int *typeLength, int keySize, int *keys)`

- **Description**: Creates a schema structure that defines the attributes of a table, including the number of attributes, their names, data types, and any key information. This schema is used to define the layout of records within a table.
- **Purpose**: To create a structured schema for a table.
- **Returns**: A pointer to the newly created schema.

`RC freeSchema(Schema *schema)`

- **Description**: Frees the memory allocated for a schema structure, ensuring that all associated resources are properly released. This prevents memory leaks after the schema is no longer needed.
- **Purpose**: To deallocate memory used by a schema.
- **Returns**: `RC_OK` if the memory is successfully freed.

Part 6: Record and Attribute Operations

This section describes the creation and deletion of records, as well as setting and retrieving attribute values within a record.

`RC createRecord(Record **record, Schema *schema)`

- **Description**: Allocates memory for a new record, ensuring that enough space is available based on the schema's size. It initializes the record structure for future data storage.
- **Purpose**: To create a new record with allocated memory.
- **Returns**: `RC_OK` if the record is successfully created, `RC_MEMORY_ALLOCATION_ERROR` if there is insufficient memory.

`RC freeRecord(Record *record)`

- **Description**: Frees the memory allocated for a record, releasing the space used by the record's data.
- **Purpose**: To prevent memory leaks by deallocating memory used by a record.
- **Returns**: `RC_OK` if the memory is successfully freed.

`RC getAttr(Record *record, Schema *schema, int attrNum, Value **value)`

- **Description**: Retrieves the value of a specific attribute in a record. It uses the schema to locate the attribute and returns its value based on its type (e.g., integer, string).
- **Purpose**: To extract an attribute's value from a record for reading or processing.
- **Returns**: `RC_OK` if the value is successfully retrieved.

`RC setAttr(Record *record, Schema *schema, int attrNum, Value *value)`

- **Description**: Updates the value of a specific attribute within a record. It uses the schema to calculate the correct position of the attribute and modifies its value based on the provided data.

- **